Linux bevezető
(jegyzet)

Bérci Norbert, Uhlár László

2015. szeptember 24-i óra anyaga

Tartalomjegyzék

1. Egy kis történelem **1**
1.1. A kezdetek . . . 1
1.2. A GNU projekt . . . 2
1.3. A Linux . . . 2

2. Programok futtatása **3**
2.1. Paraméterek . . . 4

3. Könyvtárak és elérési utak **4**

4. Környezeti változók **6**
4.1. Összefűzés . . . 7
4.2. PATH . . . 7
4.3. PS1 . . . 8
4.4. LANG . . . 9

5. Fájlok kezelése **9**
5.1. Mintaillesztés . . . 12
5.2. Egyéb speciális karakterek . . . 14

6. Átirányítás **16**

1. Egy kis történelem

1.1. A kezdetek

A számítógépek az ötvenes évektől a nyolcvanas évek elejéig a kutatók eszközei voltak és a tömegek elől gyakorlatilag el voltak zárva. Több kutató használt egy nagy gépet, a fejlesztéseiket megosztották egymással, igazi kis közösségek jöttek így létre. A kor egyik legendás gépe volt a PDP10-es, ezen az ITS nevű operációs rendszer futott, aminek továbbfejlesztésén dolgozott Richard Stallman is (lásd később). A PDP10 fejlesztését azonban a gyártója a nyolcvanas évekre abbahagyta, az intézeteknek újabb gépek után kellett nézniük. Ezek már más operációs rendszert futtattak, melyek nem voltak szabadok, már ahhoz is titoktartási szerződést kellett aláírniuk, ha egy futtatható másolatot akartak. Azaz tilos lett egymásnak segíteni, az eddig együttműködő közösségek felbomlottak, nem oszthatták meg egymással fejlesztéseiket.

[0] $Revision : 62\ Date : 2014 - 09 - 2011 : 32 : 02 + 0200(Sat, 20Sep2014)$

1.2. A GNU projekt

Richard Stallman ezt az új helyzetet nem tudta elfogadni, elhatározta egy új, teljesen nyílt operációs rendszernek a megírását: 1983 táján létrehozta a GNU projektet, hogy terveit megvalósítsa. (A GNU jelentése: GNU is not UNIX). Ekkor fogalmazta meg a GNU kiáltványt (melyet teljes terjedelmében például a http://gnu.hu oldalon olvashatunk el) és a szabad szoftverekkel kapcsolatos alapelveit:

- a program szabadon használható bármilyen célra,
- a programot bárki szabadon módosíthatja igényei szerint,
- a programot bárki továbbadhatja akár ingyen akár pénzért,
- a program módosított verzióinak meg kell felelnie ugyanezen feltételeknek.

Ezen elvek jogilag is megfelelő formába öntésének eredményeképpen jött létre egy különleges licenc, a GPL (General Public Licence) (teljes szövege magyarul szintén a http://gnu.hu oldalon olvasható) és a szabad szoftvereket támogató alapítvány, az FSF (Free Software Foundation) is. Ez utóbbi céljáról, működéséről részletesebb leírás olvasható többek között az fsf.hu oldalon.

A GNU projekt keretében számos programot kifejlesztettek, már csak egy valami hiányzott: egy olyan kernel (az operációs rendszer magja), melyen futtatni lehetne ezeket.

1.3. A Linux

1991-ben egy finn egyetemista, Linus Torvalds épp egy új projekten kezdett el dolgozni: egy új, szabad operációs rendszeren, melyben ki akarta javítani az oktatásra akkoriban előszeretettel használt MINIX operációs rendszer hibáit, hiányosságait. Azaz adott volt egy kernel (Linus munkája) alkalmazások nélkül és adott volt egy alkalmazás gyűjtemény (GNU) kernel nélkül. Nem kellett sok idő ahhoz, hogy egymásra találjanak, így született meg a Linux, aminek az előzőek miatt a legpontosabb elnevezése: GNU/Linux. Ma már a GNU programokon kívül más projektből, más licenceléssel terjesztett szoftverek is tartoz(hat)nak egy disztribúcióhoz, így ma a Linux alatt a teljes operációs rendszert értjük. Pár év, és megjelentek az első disztribúciók: a kernel és a rengeteg GNU alkalmazás közül néhány összeépítve egy jól használható rendszerré (pl.: Debian: 1993. augusztus 16.).

Egy Linux disztribúció alatt tehát egy gondosan összeválogatott, rendszermagból, felhasználói- és rendszerszintű programokból álló, szerteágazó vagy specifikus felhasználásra alkalmas operációs rendszert értünk. Egy-egy nagyobb disztribúcióban olyan sok program található, hogy nagyon ritkán van szükség külső forrásból származó programok telepítésére. Ennek az az előnye, hogy a szoftverkomponensek egymáshoz alakíthatók, az együttes installálásuk és alkalmazásuk a lehető legkevesebb mértékben vezet hibás működésre. Sőt, a szoftvercsomagok egymásra épülése is megadható, így egy szoftver installálásakor a szükséges komponensek automatikusan telepíthetők, illetve az opcionális komponensek telepítésére felhívhatja a felhasználó figyelmét. Mindezek miatt a csomagok szigorú verziószámozással vannak ellátva, és az installáláshoz *csomagkezelő*t használunk, ami a fenti függőségeket ellenőrzi és az installálás óta kiadott frissítéseket is nyomon követi.

1.3.1. Disztribúciók közötti leggyakoribb különbségek

A disztribúciókat legtöbbször az különbözteti meg, hogy milyen célközönségnek és milyen feladatra készítik őket, így mindenki megtalálhatja a neki leginkább megfelelőt. Léteznek olyanok, melyek lehetőséget nyújtanak arra, hogy szinte az összes konfigurálási lehetőséget egy grafikus felületen végezzük el és vannak olyanok is, amelyek megkövetelik, hogy a felhasználó mindent a konfigurációs állományok szerkesztésével állítson be a saját ízlésének megfelelően. Egyes disztribúciók célja, hogy mindig a lehető legfrissebb szoftvereket szállítsa, míg mások jól kitesztelt, stabil, ám emiatt kissé elavult csomagokat szállítanak. A legtöbb disztró adott közönséget céloz meg: profi vagy kezdő felhasználókat, adminisztrátorokat, „buherátorokat”, kevés memóriával rendelkező vagy csak CD-t tartalmazó gépeket stb. Néhány disztró a grafikus környezetet, míg mások inkább a karakteres konzolt támogatják.

További fontos különbség, hogy milyen csomagkezelőt használnak az adott terjesztésben. A könyvtárstruktúra általában hasonló módon van felépítve, viszont kisebb különbségek adódhatnak e tekintetben is, extrém esetekben teljesen eltérő felépítést is alkalmaznak a disztribútorok (pl.: GoboLinux). A disztrók egyik fő jellemzője az egyes programcsomagok installálásának, eltávolításának és frissítésének megkönnyítése és támogatása. A csomagkezelők a rengeteg feltelepíthető program karbantartását, frissítését, telepítését, stb. teszik könnyebbé: például a GNU/Debian 7 esetében kb. 37000 különböző program közül válogathatunk, így szinte biztosan megtaláljuk a felmerült feladataink megoldásához szükséges szoftvereket e bőséges választékban. Az egyes programok, csomagok pontos verzió számmal vannak ellátva, egy-egy program megfelelő működéséhez szükség lehet más programokra is, azaz függőségei lehetnek. Ezen függőségek (lehetőleg automatikus) feltelepítését is a csomagkezelők végzik.

Hardvertámogatás terén is adódhatnak különbségek, viszont alapvetően mind ugyanazt a kernelt használják, így elviekben ha egy disztribúció alatt egy hardver működik, akkor az bármely más, az adott architektúrát támogató disztribúció alatt is működésre bírható. Vannak céldisztribúciók is, például kifejezetten tűzfal vagy router üzemeltetésére. Megkülönböztethetjük őket az alapján is, hogy server, desktop vagy embedded felhasználásra szánják.

A disztrók nagy részének készítői komolyan veszik a biztonsági problémákat, és az ismert hibák javításait rövid időn belül elérhetővé teszik disztrójuk csomagfrissítési módszerének segítségével.

Nagy eltérések vannak a disztrók kiadásai között eltelt időben; egyes disztrók fix ciklust alkalmaznak (például 6 hónaponként egy új kiadás), más disztróknál nincs kötött kiadási ciklus. Léteznek kereskedelmi terjesztések és vállalati és otthoni / kis irodai disztribúciók is.

Nem mindegyik disztró ugyanazt a kernel verziót használja, továbbá sok disztró saját igényeinek megfelelően módosítja a hivatalosan kiadott, ún. „vanilla kernelt."

A nagyobb és ismertebb disztribúciók (a teljesség igénye nélkül):

- UHU-Linux, magyar Linux-disztribúció
- Debian GNU/Linux
- Ubuntu, Kubuntu, Xubuntu
- Mandriva
- Red Hat
- Fedora
- CentOS
- openSUSE
- Slackware
- Gentoo
- Arch Linux
- Knoppix, Damn Small Linux, Live CD-ként való futtatásra tervezve
- CrunchBang Linux

Egy kis érdekesség: http://futurist.se/gldt/wp-content/uploads/12.10/gldt1210.svg

2. Programok futtatása

Nagyon fontos, hogy a jegyzet hátralévő részében szereplő parancsokat kipróbáljuk! Erre több lehetőségünk adódik:

- egy már telepített linux verzió használatával,

- a putty programmal belépünk valamely egyetemi szerverre (users, cortex) és ott dolgozunk
- egy letöltött és futtatott live CD/DVD/pendrive használatával
  (pl.: http://live.debian.net)
- új linux telepítésével (pl.: http://debian.org)

Bármelyik módot is használjuk, a belépés után a következő(höz hasonló) promptot kell kapnunk (ha grafikus felületet használunk, a belépés után el kell indítani a `Terminal` vagy `xterm` vagy valamilyen hasonló nevű alkalmazást):

```
bercin@users:~$
```

Ez a prompt a parancsértelmező [shell] kész állapotát jelöli, kezdhetjük begépelni a parancsokat. Mi a jegyzetben a `bash` parancsértelmezőt tárgyaljuk. A legtöbb Linux disztribúcióban, illetve egyéb UNIX-okon is ezt a shellt (vagy ezzel az általunk tárgyaltak szempontjából túlnyomó többségében kompatibilis változatot) használnak.

Figyelem! A Linux (és a UNIX-ok) a kis- és nagybetűket különböző betűknek tekintik mind a fájl- és könyvtárnevek, mind a parancsok neveinek (sőt, a parancsok paramétereinek) megadásakor. Különösen ügyeljünk a helyes használatra!

2.1. Paraméterek

Az `echo` program feladata, hogy kiírja a paraméterként átadott sztringeket:

```
bercin@users:~$ echo ABCD
ABCD
bercin@users:~$ echo EFGH IJKL MNO
EFGH IJKL MNO
bercin@users:~$
```

Itt tehát az `echo` a futtatott program, ami az első esetben egyetlen paramétert kap: `ABCD`, míg a második esetben három átadott paraméter van, amik a következők: `EFGH`, `IJKL`, `MNO`. A paramétereket a legtöbb esetben szóköz karakter választja el egymástól.

A parancsokról bővebb információt a `man` (manual - kézikönyv) paranccsal lehet kérni. Például az `echo` parancsról így:

```
bercin@users:~$ man echo
```

A `man` parancs paramétere annak a parancsnak a neve, aminek a kézikönyvét meg akarjuk jeleníteni. A megjelenített kézikönyvben a kurzormozgató billentyűkkel tudunk navigálni, és a `q` billentyűvel tudunk kilépni (quit).

A programok többségének van `man` oldala, amiket a későbbiekben tárgyalt parancsok esetében is érdemes megnézni, mert a jegyzetben a parancsok lehetőségeinek csak töredékét tárgyaljuk. Magáról a parancsértelmezőről például a következő módon lehet bővebb információt szerezni:

```
bercin@users:~$ man bash
```

2.1.1. feladat. Hogyan lehet a `man` parancs kézikönyvét megnézni?

3. Könyvtárak és elérési utak

Minden futtatott program valamilyen könyvtárban fut, amit a program *aktuális könyvtár*ának [(current) working directory] nevezünk. Ez a shell esetében is így van. Az aktuális könyvtárat a `cd` (change directory) paranccsal változtathatjuk meg: paraméterként annak a könyvtárnak a nevét kell megadni, amibe be akarunk lépni.

A könyvtárak fa struktúrában ábrázolhatók, azaz egy könyvtárban több másik könyvtár (vagy fájl) lehet, amikben ismét lehetnek újabb könyvtárak (vagy fájlok). Az viszont biztos, hogy minden könyvtárnak egyetlen szülő könyvtára van. Azt az útvonalat (könyvtárak adott

sorrendjét), amellyel egy adott könyvtárhoz vagy fájlhoz eljuthatunk, a könyvtár vagy fájl *elérési útj*ának nevezzük. Az elérési útban a könyvtárakat `/` jel választja el egymástól[1].

Minden könyvtárban létezik a `.` könyvtár, ami az aktuális könyvtárat (azaz saját magát), és a `..` nevű könyvtár, ami az adott könyvtár szülő könyvtárát jelöli. Próbáljuk ki:

```
bercin@users:~$ cd .
bercin@users:~$
```

Nem szabad meglepődni, hogy nem történt semmi, mert az aktuális könyvtárból az aktuális könyvtárba lépni nyilván semmilyen változást nem okozhat. Ugyanakkor a `..` (azaz a szülő) könyvtárba lépés már nem haszontalan:

```
bercin@users:~$ cd ..
bercin@users:/home$
```

Ennek a parancsnak az eredménye alapján láthatóvá vált: a parancsértelmező eddig is kiírta, hogy éppen melyik könyvtárban vagyunk, csak erre eddig nem fordítottunk figyelmet: most, a dollárjel előtti `/home` azt jelzi, hogy az aktuális könyvtár a `/home`-ra változott. Az eddig ott szereplő `~` (hullámjel, tilde) rövidítés az alapértelmezett könyvtárunkat jelezte. Linux (UNIX) alatt alapértelmezett esetben a felhasználóknak van egy könyvtára, ahova a felhasználó írási joggal rendelkezik, és ebbe a könyvtárba kerül, amikor belép a szerverre. Ez a könyvtár a `/home/felhasználónév` (ahol `felhasználónév` a saját felhasználónk neve) amit a felhasználó *home könyvtár*ának [home directory] nevezünk. A `cd ..` parancs tehát a `/home/felhasználónév` könyvtárból ennek szülő könyvtárába, azaz a `/home` könyvtárba vitt, és a parancsértelmező ezt a változást jelezte a promptban.

Az aktuális könyvtár neve lekérdezhető a `pwd` (print working directory) parancs segítségével:

```
bercin@users:/home$ pwd
/home
bercin@users:/home$ cd ..
bercin@users:/$ pwd
/
bercin@users:/$
```

A `/` nevű könyvtár a fájlrendszer gyökér könyvtárát jelöli, azaz azt a könyvtárt, aminek a szülő könyvtára is saját maga, így ebből a könyvtárból feljebb már nem lehet lépni:

```
bercin@users:/$ pwd
/
bercin@users:/$ cd ..
bercin@users:/$ pwd
/
bercin@users:/$
```

Ha az elérési út `/` jellel kezdődik, akkor az elérési utat *teljes elérési út*nak [full path] nevezzük. A `pwd` parancs mindig az aktuális könyvtár teljes elérési útját írja ki:

```
bercin@users:/$ pwd
/
bercin@users:/$ cd home
bercin@users:/home$ pwd
/home
bercin@users:/home$ cd bercin
bercin@users:~$ pwd
/home/bercin
bercin@users:~$
```

[1] Figyelem! A könyvtár elválasztó jel Linuxban (és UNIX-ban) „jobbra dőlő” perjel, nem az, amit a Windows használ (ahol „balra dőlő” perjel választja el a könyvtárakat).

Ha az elérési út nem / jellel kezdődik, akkor is elérési útról beszélünk, de ezt az elérési utat *relatív elérési út*nak [relative path] nevezzük. A relatív elérési út azt jelenti, hogy az elérési út nem a gyökér könyvtártól, hanem az aktuális könyvtártól indul.

```
bercin@users:~$ pwd
/home/bercin
bercin@users:~$ cd /usr
bercin@users:/usr$ pwd
/usr
bercin@users:/$ cd local/bin
bercin@users:/usr/local/bin$ pwd
/usr/local/bin
bercin@users:/usr/local/bin$
```

A paraméter nélküli `cd` parancs visszavisz minket a home könyvtárunkba (bármi is az aktuális könyvtár):

```
bercin@users:/usr/local/bin$ pwd
/usr/local/bin
bercin@users:/usr/local/bin$ cd
bercin@users:~$ pwd
/home/bercin
bercin@users:~$
```

Új könyvtárat az `mkdir` paranccsal hozhatunk létre, üres könyvtárat az `rmdir` paranccsal törölhetünk:

```
bercin@users:~$ mkdir teszt
bercin@users:~$ cd teszt
bercin@users:~/teszt$ pwd
/home/bercin/teszt
bercin@users:~/teszt$ cd ..
bercin@users:~$ rmdir teszt
bercin@users:~$
```

4. Környezeti változók

A programok számára többféleképpen adhatók át adatok, aminek az egyik módja az előző példákban is látható parancssori paraméterként átadás. Egy másik módja a környezeti változókon keresztül történik, amit a programok a futásuk során lekérdezhetnek (és módosíthatnak). A környezeti változók a parancsértelmezőből történő kilépéssel törlődnek.

Egy környezeti változónak úgy lehet értéket adni, hogy a változó neve után egy egyenlőség jelet majd a beállítani kívánt értéket írjuk. Például a `PLD` változónak az `abcd` értékül adása a következő módon történik:

```
bercin@users:~$ PLD=abcd
bercin@users:~$
```

Az aktuális értéket a változónév elé tett `$` jellel lehet lekérdezni oly módon, hogy a parancssorban szereplő `$változónév` szöveget a shell a `változónév` nevű változó aktuális értékére *cseréli ki*, majd a parancssort újra értelmezi. Például az `echo $PLD` értelmezése a következőképpen történik: A shell először kicseréli a `$PLD` sztringet a PLD változó értékére (ami jelen esetben `abcd`), így eredményül az `echo abcd` parancssort kapja, amit aztán újra értelmez és végrehajt:

```
bercin@users:~$ echo $PLD
abcd
bercin@users:~$
```

Nagyon fontos ismételten hangsúlyozni, hogy a shell végzi a `$PLD` sztring kicserélését a változó tartalmára, nem pedig a futtatott program! A program már csak a kicserélt sztringet kapja meg paraméterként, mit sem sejtve arról, hogy az eredetileg mi volt: az előző példában az `echo` tehát már csak az `abcd` paramétert kapja meg (amit aztán kiír).

A shellben tárolt környezeti változók listáját aktuális értékeikkel együtt a `set` paranccsal kaphatjuk meg.

4.1. Összefűzés

Szükségünk lehet arra, hogy egy környezeti változó aktuális értéke elé és/vagy mögé egy másik sztringet is beszúrjunk. Írassuk ki két környezeti változó értékét egymáshoz fűzve:

```
bercin@users:~$ ELEJE=abcd
bercin@users:~$ VEGE=efgh
bercin@users:~$ echo $ELEJE$VEGE
abcdefgh
bercin@users:~$
```

A sikeren felbuzdulva megpróbálhatunk egy környezeti változó értékéhez közvetlenül hozzáfűzni egy sztringet:

```
bercin@users:~$ echo $ELEJEefgh

bercin@users:~$ echo $ELEJE
abcd
bercin@users:~$ echo efgh
efgh
bercin@users:~$
```

Az utasítás azért nem működik (pontosabban működik, csak nem azt az eredményt adja, amit vártunk), mert a shell nem tudja, hogy mi az `ELEJE` környezeti változó értékét és a `efgh` sztringet akartuk összefűzni, hanem az `ELEJEefgh` nevű környezeti változó értékét kérdezi le, ami üres. A megoldás, hogy a `{}` karakterek közé írva a környezeti változó nevét, a shell már pontosan meg fogja tudni határozni, hogy meddig tart a környezeti változó neve, és honnét kezdődik a parancssor többi része:

```
bercin@users:~$ echo ${ELEJE}efgh
abcdefgh
bercin@users:~$
```

Természetesen ez a jelölésmód az előző példákkal is használható:

```
bercin@users:~$ echo ${ELEJE}
abcd
bercin@users:~$ echo ${VEGE}
efgh
bercin@users:~$ echo ${ELEJE}${VEGE}
abcdefgh
bercin@users:~$
```

4.2. PATH

Magának a parancsértelmezőnek is szüksége van néhány beállításra a működéséhez, ilyen például a `PATH` nevű környezeti változó, ami megadja, hogy melyik könyvtárakban kell keresni a begépelt parancsokat. Írassuk ki a PATH változó aktuális értékét a fentebb ismertetett módon:

```
bercin@users:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games
bercin@users:~$
```

Eredményül teljes elérési utak kettősponttal elválasztott sorozatát kapjuk, ami azt mutatja, hogy a begépelt parancsok keresése *kizárólag* mely könyvtárakban történik. Esetünkben ezek a `/usr/local/bin`, `/usr/bin`, `/bin` és a `/usr/games` könyvtárak (ebben a sorrendben!). Ha a begépelt parancsot ezen könyvtárak valamelyikében megtalálja a parancsértelmező, akkor azt lefuttatja. Ha nincs ilyen fájl, akkor hibajelzést ad:

```
bercin@users:~$ svnsjkvnasjlvn
-bash: svnsjkvnasjlvn: command not found
bercin@users:~$
```

A `which` paranccsal megtudhatjuk, hogy egy (létező) program pontosan melyik, PATH-ban lévő könyvtárban található. Például az `echo` program elérési útja:

```
bercin@users:~$ which echo
/bin/echo
bercin@users:~$
```

A `which` programot bármely másik programra meghívhatjuk, akár saját magára is (ekkor azt fogja kiírni, hogy ő maga hol található):

```
bercin@users:~$ which which
/usr/bin/which
bercin@users:~$
```

4.3. PS1

A `PS1` környezeti változóval beállíthatjuk a parancsértelmező promptját. Mielőtt ezt megváltoztatnánk, mentsük el a régi értéket például a `MENTES` környezeti változóba:

```
bercin@users:~$ MENTES=$PS1
bercin@users:~$
```

A környezeti változó lekérdezését nyilván nem csak az `echo` paranccsal használhatjuk, hanem bármelyik másik paranccsal is. Idézzük fel a környezeti változó értékének a lekérdezését: először a shell kicseréli a `$változónév` kifejezést a `változónév` értékére, majd az így kapott sort újra értelmezi. Az előző példában tehát a `$PS1` helyére behelyettesítődött a `PS1` értéke, amit aztán a `MENTES` változónak adtunk értékül.

A `PS1` átállítása után az új prompttal ugyanúgy használhatjuk a parancsértelmezőt, mint eddig (ne lepődjünk meg, hogy furcsán néz ki a sor, úgy gépeljük be a parancsainkat, mintha az előzőekben megszokott prompt lenne!):

```
bercin@users:~$ PS1=_ez_az_en_promptom_
_ez_az_en_promptom_echo szia
szia
_ez_az_en_promptom_which echo
/bin/echo
_ez_az_en_promptom_
```

Nézzük meg, hogyan állíthatjuk vissza az eredeti állapotot:

```
_ez_az_en_promptom_PS1=$MENTES
bercin@users:~$
```

4.3.1. feladat. Mi a `PS1` (vagy a `MENTES`) tartalma?

A `PS1` beállításáról a későbbiekben lesz még szó bővebben.

4.4. LANG

A `LANG` környezeti változó beállításával átállíthatjuk a területi beállításokat, így például megváltoztathatjuk a parancsok által kiírt üzenetek nyelvét. Értékül a használni kívánt nyelv kétbetűs kódját kisbetűkkel és az ország kétbetűs kódját nagybetűkkel kell megadni, aláhúzásjellel elválasztva:

```
bercin@users:~$ svnsjkvnasjlvn
-bash: svnsjkvnasjlvn: command not found
bercin@users:~$ LANG=hu_HU
bercin@users:~$ svnsjkvnasjlvn
-bash: svnsjkvnasjlvn: parancs nem található
bercin@users:~$
```

Ha a magyarul kiírt hibaüzenetben az ékezetes betűk helyén kérdőjel, vagy egyéb más „furcsa" karakter szerepel, akkor a `hu_HU` helyett próbáljuk ki a `hu_HU.UTF8` értéket. Az UTF-8 karakterkódolásról a későbbiekben lesz részletesen szó.

4.4.1. feladat. Hogyan tudjuk a beállítás előtt elmenteni és az átállítás után az eredetire visszaállítani a `LANG` környezeti változót?

4.4.2. feladat. Hogyan tudjuk átállítani a területi beállításokat a Németországban használt németre?

4.4.3. feladat. Hogyan tudjuk átállítani a területi beállításokat az USA (délnyugati részén gyakran) használt spanyolra?

5. Fájlok kezelése

Az `ls` paranccsal kilistázható az aktuális könyvtár tartalma (ha nem adunk meg paramétert) vagy bármely más könyvtár tartalma (ha megadunk egy elérési utat):

```
bercin@users:~$ ls
public_html  teszt  ZHk
bercin@users:~$ ls /bin
bash          dd             lessecho        nisdomainname  tar
bunzip2       df             lessfile        pidof          tempfile
bzcat         dir            lesskey         ping           touch
bzcmp         dmesg          lesspipe        ping6          true
bzdiff        dnsdomainname  ln              ps             umount
bzegrep       domainname     login           pwd            uname
bzexe         echo           ls              rbash          uncompress
bzfgrep       ed             lsmod           readlink       vdir
bzgrep        egrep          mkdir           rm             which
bzip2         false          mknod           rmdir          ypdomainname
bzip2recover  fgrep          mktemp          rnano          zcat
bzless        fuser          more            run-parts      zcmp
bzmore        getfacl        mount           rzsh           zdiff
cat           grep           mountpoint      sed            zegrep
chacl         gunzip         mt              setfacl        zfgrep
chgrp         gzexe          mt-gnu          sh             zforce
chmod         gzip           mv              sh.distrib     zgrep
chown         hostname       nano            sleep          zless
cp            ip             nc              stty           zmore
cpio          kill           nc.traditional  su             znew
dash          ksh            netcat          sync           zsh
date          less           netstat         tailf          zsh4
bercin@users:~$
```

Ebben a példában az aktuális könyvtár elemeit (első-második sor) és a `/bin` könyvtár elemeit listáztuk ki (harmadik sortól az utolsóig).

Egy újonnan létrehozott könyvtár is tartalmazza a `.` és `..` könyvtárakat, de ezeket az `ls` alapértelmezetten nem mutatja:

```
bercin@users:~$ mkdir teszt
bercin@users:~$ cd teszt
bercin@users:~/teszt$ ls
bercin@users:~/teszt$
```

Linuxban (és UNIX-ban) a ponttal kezdődő könyvtárnevek és fájlnevek rejtettek, azaz az `ls` alapértelmezetten nem listázza ki ezeket. Ha az `ls` parancsot a `-a` paraméterrel hívjuk meg, akkor már megmutatja a ponttal kezdődő könyvtárakat illetve állományokat is:

```
bercin@users:~/teszt$ ls -a
.  ..
bercin@users:~/teszt$
```

Természetesen a `.` és `..` könyvtár minden könyvtárban megtalálható, bármelyiket is listázzuk ki:

```
bercin@users:~/teszt$ cd
bercin@users:~$ ls
public_html  teszt  ZHk
bercin@users:~$ ls -a
.   .bash_history  .bashrc  .lesshst  .profile     .ssh   ZHk
..  .bash_logout   .gnupg   .mc       public_html  teszt
bercin@users:~$
```

Jól látható ezen a példán, hogy a `.` és `..` könyvtárak mellett egy felhasználó home könyvtára általában sok más, ponttal kezdődő fájlt is tartalmaz, amik az előzőek alapján szintén rejtettek.

Linux (UNIX) esetében a felhasználó által futtatott programok beállításait általában ponttal kezdődő fájlnevű állományban tároljuk. A programjaink config fájljai tehát azért a felhasználó home könyvtárában vannak, mert így lehetővé válik, hogy felhasználónként más és más beállítások legyenek. Szükségszerű is itt tárolni a config állományokat, hiszen ezek máshol nem biztos, hogy tárolhatók, mert általános esetben a felhasználónak csak a home könyvtárában van írási joga.

Fájlokat másolni a `cp` (copy) paranccsal lehet, aminek első paraméterként meg kell adni azt az elérési utat, amit másolni szeretnénk, második paraméterként pedig azt az elérési utat, ahova másolni szeretnénk. Például a már jól ismert `echo` programot másoljuk a saját home könyvtárunkba `masik_echo` néven:

```
bercin@users:~$ cp /bin/echo masik_echo
bercin@users:~$ ls
masik_echo  public_html  teszt  ZHk
bercin@users:~$
```

Hiába létezik az aktuális könyvtárban a `masik_echo` fájl, ha a megpróbáljuk lefuttatni, hibajelzést kapunk:

```
bercin@users:~$ masik_echo
-bash: masik_echo: command not found
bercin@users:~$
```

Ennek magyarázata, hogy a shell kizárólag a `PATH` környezeti változóban felsorolt könyvtárakban keres, és ezek között nem található meg az aktuális könyvtár:

```
bercin@users:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games
bercin@users:~$
```

A problémát egyszerűen orvosolhatjuk: a parancsnév helyett adjuk meg a futtatandó parancs elérési útját, így a shellnek nem kell a PATH-ban lévő könyvtárakban keresnie, hanem közvetlenül a megadott fájlt próbálja meg futtatni. Elérési útként megadható teljes elérési út:

```
bercin@users:~$ /home/bercin/masik_echo szia
szia
bercin@users:~$
```

És megadható relatív elérési út is:

```
bercin@users:~$ ./masik_echo szia
szia
bercin@users:~$
```

A shell úgy tudja megkülönböztetni, hogy relatív elérési utat vagy elérési út nélküli parancsnevet adtunk meg, hogy ha a parancsban nincsen / karakter, akkor parancsnévről van szó (és a PATH-ban megadott könyvtárakban keresi a parancsot), de ha van benne / karakter, akkor elérési útról van szó (sőt, ha az elérési út / jellel kezdődik, akkor abszolút elérési út, azaz a gyökér könyvtártól indul, ha nem / jellel kezdődik, akkor relatív, azaz az aktuális könyvtártól indul).

Pontosan azért volt tehát szükség az előző példában a ./ szerepeltetésére, hogy a shell tudja, elérési úttal megadott parancsot akarunk futtatni, és a . könyvtár megadásával ez pontosan azt jelenti, hogy az aktuális könyvtárban lévő fájlról van szó.

Erre a problémára az előző megoldáson kívül másik két megoldás is létezik: az egyik, hogy az aktuális könyvtárat (tehát a . könyvtárat) is beillesztjük a PATH listába, a másik, hogy létrehozunk egy saját könyvtárat a home könyvtárunkban, ahol a saját futtatható állományainkat fogjuk tárolni. Az első megoldás biztonsági kockázatokat rejt (ezért nem is keres a shell az aktuális könyvtárban), így válasszuk a második megoldást. Hívjuk ezt a könyvtárat bin-nek:

```
bercin@users:~$ mkdir bin
bercin@users:~$ PATH=$PATH:~/bin
bercin@users:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/games:/home/bercin/bin
bercin@users:~$
```

Létrehoztuk tehát a bin könyvtárat a home könyvtárunkban, majd a PATH környezeti változó végéhez hozzáfűztük ennek az újonnan létrehozott könyvtárnak az elérési útját. Részletesebben itt két csere is történt: a $PATH:~/bin sztringet a shell értelmezte és a $PATH sztringet kicserélte a PATH környezeti változó aktuális értékére, majd a ~ jelet is kicserélte a felhasználó home könyvtárának elérési útjára, így eredményül (ebben az esetben) a

```
/usr/local/bin:/usr/bin:/bin:/usr/games:/home/bercin/bin
```

értéket kapta. A végrehajtási fázisban tehát az történt, mintha a

```
PATH=/usr/local/bin:/usr/bin:/bin:/usr/games:/home/bercin/bin
```

parancsot írtuk volna be, ezt bizonyítja a PATH környezeti változó értéke kiíratásának eredménye.

A feladat megoldásához már csak egy lépés van hátra, az aktuális könyvtárban lévő masik_echo fájlt át kell mozgatni a bin könyvtárba, amit a mv (move) parancs segítségével tehetünk meg:

```
bercin@users:~$ ls
masik_echo  public_html  teszt  ZHk
bercin@users:~$ ls -a bin
.  ..
bercin@users:~$ mv masik_echo bin
bercin@users:~$ ls -a bin
.  ..  masik_echo
bercin@users:~$ ls
bin  public_html  teszt  ZHk
bercin@users:~$
```

Próbáljuk ki, hogy most már működik-e a `masik_echo` program indítása, holott az nincs is az aktuális könyvtárban:

```
bercin@users:~$ ls
bin  public_html  teszt  ZHk
bercin@users:~$ masik_echo szia
szia
bercin@users:~$
```

Az `mv` parancs az átmozgatás mellett átnevezésre is használható: gondoljunk bele, ha átmozgatok egy fájlt az aktuális könyvtárból az aktuális könyvtárba csak másik néven, akkor valójában átnevezést hajtottam végre. Emiatt nincs külön parancs átnevezésre a Linuxban (és a UNIX-okban).

Ha a `cp` vagy `mv` parancsnak második paraméterként (tehát a cél elérési útként) könyvtárat adunk meg, akkor abba a könyvtárba ugyanolyan néven fog másolódni (`cp` esetében) illetve átmozgatódni (`mv` esetében) a forrásként megadott fájl. Természetesen több fájlt is másolhatunk egy paranccsal, ha felsoroljuk az összes másolandó fájlt, majd az utolsó paraméterként megadjuk azt a könyvtárat, ahova azokat másolni szeretnénk. Ekkor az utolsó paraméter csak könyvtár lehet!

```
bercin@users:~$ ls bin
masik_echo
bercin@users:~$ cp /bin/mv /bin/cp /bin/mkdir /bin/rmdir bin
bercin@users:~$ ls bin
cp  masik_echo  mkdir  mv  rmdir
bercin@users:~$
```

Fájlok törlésére az `rm` parancs szolgál, paraméterekként meg kell adni a törlendő fájlok elérési útjait. Például az imént a `bin` könyvtárba másolt állományok közül néhány törlése:

```
bercin@users:~$ ls bin
cp  masik_echo  mkdir  mv  rmdir
bercin@users:~$ rm bin/mv bin/mkdir bin/cp
bercin@users:~$ ls bin
masik_echo  rmdir
bercin@users:~$
```

Természetesen ha egy könyvtár összes fájlját törölni szeretnénk, nem kell azokat egyesével felsorolni, hanem megadhatjuk a könyvtárnevet is az `rm` parancsnak:

```
bercin@users:~$ rm bin
rm: cannot remove ‘bin’: Is a directory
bercin@users:~$
```

Mivel alapértelmezetten az `rm` parancs fájlok törlésére szolgál, külön paraméter megadása szükséges, hogy könyvtárat töröljünk: `-r` (rekurzív törlés)

```
bercin@users:~$ rm -r bin
bercin@users:~$
```

Figyelem! Linuxban (és UNIX-ban) a törlés általában visszafordíthatatlan művelet! Nincs „Trash” vagy „Kuka”, amibe ideiglenesen átkerülnek a törölt állományok! Az `rm` parancs kiadásakor azok azonnal törlődnek.

5.1. Mintaillesztés

A shell nem csak a környezeti változók értékének lekérdezésére használt `$` jelet értelmezi speciálisan, hanem lehetőség van a fájlok neveinek mintaillesztésére, azaz a létező fájlnevek közül adott mintának megfelelők kiválasztására. A shell által erre a célra használt speciális karakterek és jelentéseik az 1. táblázatban láthatóak.

illesztő karakter	karakterek, amelyek megfelelnek a mintának
*	bármely karakter akárhányszor (nulla alkalommal is!)
?	bármely karakter pontosan egy alkalommal
[karakterek]	karakterek közül bármelyik pontosan egy alkalommal

1. táblázat. A shell mintaillesztő karakterei

A mintaillesztést a `/bin` könyvtárban fogjuk szemléltetni. Álljon itt egy lista a benne található fájlokról annak érdekében, hogy a lentebbi példákat a szerverre való belépés nélkül is meg lehessen érteni.

```
bercin@users:~$ cd /bin
bercin@users:/bin$ ls
bash          dd              lessecho        nisdomainname  tar
bunzip2       df              lessfile        pidof          tempfile
bzcat         dir             lesskey         ping           touch
bzcmp         dmesg           lesspipe        ping6          true
bzdiff        dnsdomainname   ln              ps             umount
bzegrep       domainname      login           pwd            uname
bzexe         echo            ls              rbash          uncompress
bzfgrep       ed              lsmod           readlink       vdir
bzgrep        egrep           mkdir           rm             which
bzip2         false           mknod           rmdir          ypdomainname
bzip2recover  fgrep           mktemp          rnano          zcat
bzless        fuser           more            run-parts      zcmp
bzmore        getfacl         mount           rzsh           zdiff
cat           grep            mountpoint      sed            zegrep
chacl         gunzip          mt              setfacl        zfgrep
chgrp         gzexe           mt-gnu          sh             zforce
chmod         gzip            mv              sh.distrib     zgrep
chown         hostname        nano            sleep          zless
cp            ip              nc              stty           zmore
cpio          kill            nc.traditional  su             znew
dash          ksh             netcat          sync           zsh
date          less            netstat         tailf          zsh4
bercin@users:/bin$
```

Írassuk ki a `c` karakterrel kezdődő fájlokat:

```
bercin@users:/bin$ ls c*
cat  chacl  chgrp  chmod  chown  cp  cpio
bercin@users:/bin$
```

A `c*` mintának tehát megfelel minden olyan fájl, ami `c` karakterrel kezdődik és utána bármely karakterből bárhány szerepel. A `*` karakter azonban nem csak a minta végén lehet, sőt, nem csak egy szerepelhet belőle egy mintában:

```
bercin@users:/bin$ ls *l*
bzless  getfacl  lessecho  lesspipe  ls              readlink  tailf
chacl   kill     lessfile  ln        lsmod           setfacl   tempfile
false   less     lesskey   login     nc.traditional  sleep     zless
bercin@users:/bin$
```

A `*l*` mintának megfelel az összes olyan fájl, amiben legalább egy `l` betű szerepel. Hasonlóan egyszerű kilistázni azokat a fájlokat, amelyekben legalább egy `l` és utána legalább egy `s` betű szerepel:

```
bercin@users:/bin$ ls *l*s*
```

```
bzless  less      lessfile  lesspipe  lsmod
false   lessecho  lesskey   ls        zless
bercin@users:/bin$
```

Figyelem! A karakterek sorrendje fontos! Az előző minta nem ugyanaz, mint a `*s*l*`:

```
bercin@users:/bin$ ls *s*l*
lessfile  setfacl  sleep
bercin@users:/bin$
```

5.1.1. példa. Listázzuk ki azokat a fájlokat, amelyek második karaktere `e`:

```
bercin@users:/bin$ ls ?e*
getfacl  lessecho  lesskey   netcat   readlink  setfacl   zegrep
less     lessfile  lesspipe  netstat  sed       tempfile
bercin@users:/bin$
```

5.1.2. példa. Listázzuk ki azokat a fájlokat, amelyek második karaktere `e` és a hatodik karaktere `c`:

```
bercin@users:/bin$ ls ?e???c*
getfacl  lessecho  setfacl
bercin@users:/bin$
```

5.1.3. példa. Listázzuk ki azokat a fájlokat, amelyek pontosan három karakterből állnak:

```
bercin@users:/bin$ ls ???
cat  dir  ksh  pwd  sed  tar  zsh
bercin@users:/bin$
```

5.1.4. példa. Listázzuk ki azokat a fájlokat, amelyek az `abcd` karakterek közül valamelyikkel kezdődnek és utolsó karakterük `e`:

```
bercin@users:/bin$ ls [acbd]*e
bzexe  bzmore  date  dnsdomainname  domainname
bercin@users:/bin$
```

A mintaillesztés természetesen nem csak a fájlok kilistázásánál, hanem másolásnál (`cp`), átmozgatásnál (`mv`), törlésnél (`rm`) is hasznos, sőt – mivel a behelyettesítést a shell végzi – bármely más program is felhasználhatja ezt a funkciót, ha fájlok neveit várja paraméterként.

5.2. Egyéb speciális karakterek

Eddig elhallgattuk, de a szemfüles olvasó már találkozhatott azzal a problémával, hogy (például könyvtár létrehozásánál) a fentiek alapján nem tud szóköz karaktert tartalmazó nevet használni, hiszen a szóköz karakter paraméter elválasztó karakter, így ha megpróbálunk egy `Kedves Hallgatok` nevű könyvtárat létrehozni, akkor azt két külön könyvtárként fogja a shell létrehozni:

```
bercin@users:~$ mkdir Kedves Hallgatok
bercin@users:~$ ls
bin  Hallgatok  Kedves  public_html  teszt  ZHk
bercin@users:~$
```

Hasonló a probléma a mintaillesztő karakterek használatával is:

```
bercin@users:~$ mkdir ***K***
mkdir: cannot create directory ‘Kedves’: File exists
bercin@users:~$ ls
bin  Hallgatok  Kedves  public_html  teszt  ZHk
bercin@users:~$
```

A hibaüzenet magyarázata, hogy a shell a `***K***` mintát behelyettesíti az aktuális könyvtárban található `Kedves` fájlnévvel (ez létező név, hiszen éppen az előbb hoztuk létre ezt a könyvtárat), így az `mkdir` parancs a `Kedves` paramétert kapja, ami nyilván hibához vezet, hiszen nem hozható létre egy már meglévő névvel új könyvtár.

A problémát úgy orvosolhatjuk, hogy a speciális karakterek speciális jelentését kikapcsoljuk, azaz a kikapcsolás után a karakterek már önmagukat jelentik: a `$` karaktert nem próbálja meg a shell változó értékére cserélni, a mintaillesztő karaktereket nem próbálja meg fájlnevekre illeszteni, stb. A speciális jelentés kikapcsolására a `\` karakter használandó. Figyelem! Itt „balra dőlő" perjelről van szó, nem a könyvtárakat elválasztó „jobbra dőlő" perjelről!

Egy szóközt tartalmazó könyvtár létrehozására használható tehát a következő módszer:

```
bercin@users:~$ mkdir Ez\ itt\ egy\ teljes\ mondat
bercin@users:~$ ls
bin  Ez itt egy teljes mondat  Hallgatok  Kedves  public_html  teszt  ZHk
bercin@users:~$
```

Egy kicsit problémás minden speciális karakter elé beszúrni a `\` jelet, de ez a probléma is meg van oldva: ha több karakter speciális jelentését akarjuk kikapcsolni, akkor a sztringet `'` jelek közé kell tenni:

```
bercin@users:~$ mkdir 'Ez itt egy masik hosszu nevu konyvtar'
bercin@users:~$ ls
bin                                  Hallgatok    teszt
Ez itt egy masik hosszu nevu konyvtar  Kedves       ZHk
Ez itt egy teljes mondat             public_html
bercin@users:~$
```

Néha azonban mégis szükség lenne arra, hogy környezeti változókat is tudjunk idézőjelek között megadott sztringekben szerepeltetni:

```
bercin@users:~$ szamlaszam=123456789
bercin@users:~$ mkdir 'A bankszamlam szama: $szamlaszam'
bercin@users:~$ ls
A bankszamlam szama: $szamlaszam       Ez itt egy teljes mondat  public_html
bin                                  Hallgatok                 teszt
Ez itt egy masik hosszu nevu konyvtar  Kedves                    ZHk
bercin@users:~$
```

A könyvtárnévben nem helyettesítődött be a `szamlaszam` környezeti változó értéke. Megoldás: ha a sztinget `"` jelek közé tesszük, akkor a shell a `$` értelmezését továbbra is meg fogja tenni:

```
bercin@users:~$ mkdir "A bankszamlam szama: $szamlaszam"
bercin@users:~$ ls
A bankszamlam szama: 123456789         Ez itt egy teljes mondat  teszt
A bankszamlam szama: $szamlaszam       Hallgatok                 ZHk
bin                                  Kedves
Ez itt egy masik hosszu nevu konyvtar  public_html
bercin@users:~$
```

Természetesen magának a környezeti változónak is adható a fenti módon szóközt (vagy egyéb más speciális karaktert) tartalmazó érték:

```
bercin@users:~$ telefonszam="+36 12 345 6789"
bercin@users:~$ mkdir "A telefonszamom: $telefonszam"
bercin@users:~$ ls
A bankszamlam szama: 123456789         Hallgatok
A bankszamlam szama: $szamlaszam       Kedves
A telefonszamom: +36 12 345 6789       public_html
bin                                  teszt
```

```
Ez itt egy masik hosszu nevu konyvtar  ZHk
Ez itt egy teljes mondat
bercin@users:~$
```

A promptot most már átállíthatjuk úgy, hogy az sokkal szebben nézzen ki, mint az első példában (lásd a 4.3. részt):

```
bercin@users:~$ PS1="Ez az en promptom: "
Ez az en promptom: echo szia
szia
Ez az en promptom: ls
A bankszamlam szama: 123456789         Hallgatok
A bankszamlam szama: $szamlaszam       Kedves
A telefonszamom: +36 12 345 6789       public_html
bin                                    teszt
Ez itt egy masik hosszu nevu konyvtar  ZHk
Ez itt egy teljes mondat
Ez az en promptom:
```

A prompt beállításakor használhatunk speciális értékeket is, amiket a shell értelmezni fog, és a megfelelő értékre cserél. Ezek közül a leggyakrabban használtak a 2. táblázatban láthatóak.

speciális karakter	karakterek, amelyek megfelelnek a mintának
`\u`	felhasználónév
`\h`	a gép neve, amire be vagyunk jelentkezve
`\H`	a gép teljes neve, amire be vagyunk jelentkezve
`\w`	az aktuális könyvtár teljes elérési úttal
`\W`	az aktuális könyvtár elérési útjának utolsó eleme
`\$`	egyszerű felhasználó esetében `$` jel, rendszergazda esetében `#` jel

2. táblázat. Prompt beállításához használható legfontosabb speciális karakterek

Ahogy a táblázatból látszik, a `\` karakter egyes esetekben speciális értelmezésű karakterek speciális értelmezésének kikapcsolására, más esetekben normál értelmezésű karakterek speciális értelmezésének bekapcsolására szolgál. Figyelem! A táblázatban szereplő karakterek a prompt beállításában értelmezettek csak a táblázat szerint!

5.2.1. példa. A prompt

```
felhasználó: <felhasználó> gép: <gép> könyvtár: <könyvtár> $
```

-ra történő beállítása a következő módon végezhető el:

```
bercin@users:~$ PS1="felhasználó: \u gép: \h könyvtár: \w $ "
felhasználó: bercin gép: www-users könyvtár: ~ $ echo szia
szia
felhasználó: bercin gép: www-users könyvtár: ~ $
```

6. Átirányítás

Linuxban (UNIX-ban) minden elindított programnak az induláskor három különböző ki-bemeneti (I/O) csatornája létezik: a 0 számmal, a C nyelvben stdin-nel, a C++ nyelvben cin-nel jelölt *sztenderd bemenet*, az 1 számmal, a C nyelvben stdout-tal, a C++ nyelvben cout-tal jelölt *sztenderd kimenet* és a 2 számmal, a C nyelvben stderr-rel, a C++ nyelvben cerr-rel jelölt *hiba kimenet*. Ezeket a csatornákat át lehet irányítani. A sztenderd kimenetet átirányíthatjuk egy fájlba a parancs után `>` jelet végül a fájl nevét megadva:

```
bercin@users:~$ ls /bin/c*
/bin/cat    /bin/chgrp  /bin/chown  /bin/cpio
/bin/chacl  /bin/chmod  /bin/cp
bercin@users:~$ ls /bin/c* > kimenet
bercin@users:~$
```

A `>` speciális karakter az `ls` parancs kimenetét a `kimenet` fájlba irányította át (létrehozva a fájlt, ha az addig nem létezett), emiatt nem látszik a parancs eredménye a képernyőn. Egy fájl tartalmát a `cat` paranccsal írathatjuk ki. Írassuk ki a `kimenet` fájl tartalmát:

```
bercin@users:~$ cat kimenet
/bin/cat
/bin/chacl
/bin/chgrp
/bin/chmod
/bin/chown
/bin/cp
/bin/cpio
bercin@users:~$
```

Egy parancs nem csak a sztenderd kimenetére írhat ki üzeneteket, hanem a hiba kimenetén is megjeleníthet szöveget: ehhez a következő példában a c-vel kezdődő fájlokat és a `asadadads` nevű fájlt is megpróbáljuk kilistázni, de az utóbbi nem létezik:

```
bercin@users:~$ ls asadadads /bin/c*
ls: cannot access asadadads: No such file or directory
/bin/cat    /bin/chgrp  /bin/chown  /bin/cpio
/bin/chacl  /bin/chmod  /bin/cp
bercin@users:~$
```

A hiba kimenet átirányítására a `2>` karaktereket kell használnunk:

```
bercin@users:~$ ls asadadads /bin/c* 2> kimenet
/bin/cat    /bin/chgrp  /bin/chown  /bin/cpio
/bin/chacl  /bin/chmod  /bin/cp
bercin@users:~$ cat kimenet
ls: cannot access asadadads: No such file or directory
bercin@users:~$
```

Jól látható, hogy a sztenderd kimenet (átirányítás hiányában) továbbra is megjelent a képernyőn, de a hibaüzenetet már a fájlban tároltuk el. Ha mind a hiba, mind a sztenderd kimenetet át akarjuk irányítani, akkor az előzőek természetesen egymás után is alkalmazhatók:

```
bercin@users:~$ ls asadadads /bin/c* > kimenet.sima 2> kimenet.hiba
bercin@users:~$ cat kimenet.sima
/bin/cat
/bin/chacl
/bin/chgrp
/bin/chmod
/bin/chown
/bin/cp
/bin/cpio
bercin@users:~$ cat kimenet.hiba
ls: cannot access asadadads: No such file or directory
bercin@users:~$
```

Ha a két kimenetet ugyanabba a fájlba akarjuk átirányítani akkor arra a következő módszert kell használni:

```
bercin@users:~$ ls asadadads /bin/c* > kimenet 2>&1
bercin@users:~$ cat kimenet
ls: cannot access asadadads: No such file or directory
/bin/cat
/bin/chacl
/bin/chgrp
/bin/chmod
/bin/chown
/bin/cp
/bin/cpio
bercin@users:~$
```

A `2>&1` azt jelenti, hogy a sztenderd error kimenetet (2) ugyanoda akarjuk átirányítani, ahova a sztenderd kimenet (1) aktuálisan irányítva van a feldolgozás pillanatában.

6.0.1. feladat. Mi a különbség az alábbiak között?
`parancs 2>&1 > fajlnev`
`parancs > fajlnev 2>&1`
Próbáljuk ki, és értelmezzük az eredményt!

Az előzőekben a kimenetet úgy irányítottuk át, hogy ha a fájl nem létezett, akkor azt a shell létrehozta, ha létezett, akkor törölte annak tartalmát, és ezután írta bele az átirányítás eredményét. Ha arra van szükségünk, hogy a fájl tartalmát ne törölje, hanem a már meglévő fájl végéhez írja hozzá az átirányítás tartalmát, akkor a `>>` karaktereket kell használni:

```
bercin@users:~$ ls /bin/cp* > kimenet
bercin@users:~$ cat kimenet
/bin/cp
/bin/cpio
bercin@users:~$ ls /bin/ch* > kimenet
bercin@users:~$ cat kimenet
/bin/chacl
/bin/chgrp
/bin/chmod
/bin/chown
bercin@users:~$ ls /bin/ca* >> kimenet
bercin@users:~$ cat kimenet
/bin/chacl
/bin/chgrp
/bin/chmod
/bin/chown
/bin/cat
bercin@users:~$
```