Raffai Gábor István
alias Glindorf

Bash Shell

Programozás

basic2bash

BASH SHELL PROGRAMOZÁS (basic2bash)

Készítette: Raffai Gábor István alias Glindorf
Kelt: Kecskemét, 2003.04.18.
(frissitve: 2003.04.25.)

Hiba bejelentés:
Mail: rgi@vipmail.hu glindorf@mailbox.hu
Mobil: 70/259-0425
ICQ: 108767553

Tartalom

Bevezetés

Üdvözlöm a kedves érdeklődőt, aki felütötte ezen az oldalakat.

Ez az írás a Bash shell scriptek írásának alapjait igyekszik bemutatni, illetve segít azt elsajátítani. Kicsit fellengzősen úgy is utalhatunk rá, mint shell programozásra. Bár nyilvánvalóan a lehetőségek terén, a shell messze elmarad a valódi program nyelvektől, még is nagyon hatékony eszköz tud lenni a kezünkben. Az, hogy milyen mértékben az tulajdonképpen rajtunk múlik. Egyrészt, hogy milyen mélységeiben ismerjük meg a bash shell-t, másrészt pedig, hogy mennyi parancssoros programot, ismerünk meg és tanulunk meg használni. Hiszen a shell scripteknek egy jellemzője, hogy a shell-en kívül különféle egyéb programokat hívunk, hívhatunk meg belőle.

Természetesen az írás messze nem törekszik teljességre. Célja, egy jó kezdethez segíteni a felhasználót, másrészt a windowshoz szokott személyeknek bepillantást engedni a linux lelkivilágába. Nem titkolt célja az írásnak, hogy felkeltse az érdeklődést és motivációt is adjon, az alapvető parancsok és konzolos programok megismeréséhez. Hogy a kezdő, de érdeklődő felhasználót, mintegy átvezesse a grafikus felület burkán és bátorságot, illetve önbizalmat adjon a konzol használatához, a config fájlok és scriptek átszerkesztéséhez, programok forrásból történő forgatásához, stb. Azaz, hogy a felhasználót, kontaktusba hozza a linux valódi "énjével".

Ez a dokumentum, az egészen kezdő felhasználókhoz is szól, akik még sohasem programoztak más nyelven. (Pl.: basic, pascal, c, stb.) De az írás nem óhajt programozástechnikai fogásokkal foglalkozni. Ebből is "csak" az alapokat adja meg.

A részeket érdemes elejétől fogva, sorba olvasni, mert fokozatosan egyre több mindent bíz a felhasználó önálló gondolkodására, már megszerzett ismereteire, azaz az anyag feltételezi az előző részek elolvasottságát és megértését. Éppen ezért, a példákat és a scripteket, érdemes végigcsinálni, elkészíteni. Egy-egy rész után, akkor jó ha tovább megyünk, ha azt, valamint az ott szereplő parancsokat és scriptek működését megértettük. A későbbi részekben egyre több információ a példa scriptek kommentjeiben van megadva illetve az utolsó részekben néhol, csak egy rövid összefoglaló leírás van a script működéséről. De aki a fentiek szerint végig követti az anyagot, annak ez nem fog problémát okozni, sőt, így jobban áttekinthetők számára a példák és a scriptek.

A példák és a példa scriptek csak az adott anyag megértését szolgálják. Nyilvánvalóan egy-egy feladatot sokféleképpen, más parancsokkal, rövidebb kóddal is meg lehet oldani. Ez is egy nagyszerű lehetőség az önképzésre és fejlődésre. Mivel a bash-script, nem más, mint parancssori utasítások sorozata, ezért néhány egyszerűbb esetben elég csak a konzolon, parancssorba írni a példát. Ezt a módszert, főleg eleinte használni fogjuk, mivel így egyszerűbben és gyorsabban követhető az anyag. Ezeknél a példáknál, egy "$" jel látható a sor elején, mely a promptot jelenti, vagyis azt nem kell a terminálba begépelni. Amelyik sor pedig az előzőleg begépelt parancs eredményét tartalmazza, ott nincs

előtte "$" jel. Ezzel lehet őket jól megkülönböztetni. Ahol script forrása szerepel, azt két szaggatott vonalakat tartalmazó sor közé helyeztem. Gyakran fogunk találkozni a scriptekben, olyan hosszú parancssorokkal, sorozatokkal, amelyek nem férnek el a könyv egy sorába. Ezek az értelmezést megnehezítik, mert nem biztos, hogy az ember rögtön látja, új sorról, van-e szó, vagy az előző folytatódik. Ezért, a „\” jelet használom a scriptekben. Ez mintegy semmissé teszi a sortörést. (Levédi a sorvégjelet.) Ha ilyen sorokat másolunk ki a dokumentumból, azok helyesen fognak végrehajtódni. Mert a shell számára, ezek egy sornak számítanak.

A scripteket bármilyen text fájlt kezelő szövegszerkesztővel elő lehet állítani, de erre a célra érdemesebb egyszerű editort, vagy direkt erre a célra kifejlesztett editort használni. A legegyszerűbb az mc editorát használni (mcedit). Ha már a fájl elején megnyitáskor szerepel a "#!/bin/bash" bejegyzés, akkor a bash shell szabályainak megfelelően emeli ki a script egyes elemeit. Ha inkább grafikus felületűt szeretnénk használni, akkor a kwrite is tudja a bash-nak megfelelően kiemelni az elemeket. Egy kisebb és gyorsabb lehetőség a nedit, ami szintén tudja színezni a szöveget a bash szabályinak megfelelően. Érdekes lehetőség még a kate, amely a kwrite-hez hasonló, de könnyű vele egyszerre több fájlt szerkeszteni, kezelni, a képernyőt meg tudja osztani két szerkesztendő fájl között, fájl csoportokat azaz fájllistákat is is lehet vele menteni, megnyitni, valamint a kate-n belül lehet nyitni egy parancsértelmezőt, azaz terminált, ami aktuális könyvtára dinamikusan, az éppen kiválasztott dokumentum könyvtárára vált.

A dokumentum egyes részeinek végén egy-egy Xdialog-os példa szerepel. Mi is azaz Xdialog ? Egy grafikus dialógus ablakokat megjelenítő program, amely scriptekből hívhatók meg. A script beli hívásakor megadott kapcsolók, paraméterek, argumentumok határozzák meg a működését és a típusától függően értékeket is képes visszaadni. Aki foglalkozott windows API-val, Visual Basic-el, Delphi-vel, Kilyx, vagy bármilyen egyéb hasonló vizuális fejlesztőeszközzel, akkor azok grafikus építőelemeire gondoljon. Természetesen összesem hasonlíthatóan kevesebb lehetőséggel. Az Xdialog-al, könnyen és gyorsan készíthetünk, "látványos" és felhasználóbarát scripteket. Az alábbi oldalakról lehet letölteni.
http://xdialog.dyns.net http://freshmeat.net/projects/xdialog

A dokumentumban szereplő példascriptek, letölthetőek a http://glindorf.fw.hu , Letöltések > Shell Programozás dokumentum - scriptek szekcióból. Illetve az oldalon megtalálható ennek a dokumentumnak a legfrisebb változata, txt és pdf formátumban is.

Szívesen veszek a dokumentummal kapcsolatos mindenféle visszajelzést, kritikát. Az alábbi címeken vagyok elérhető:
Raffai Gábor István:
Mail: rgi@vipmail.hu glindorf@mailbox.hu
ICQ: 108767553
Web: http://glindorf.fw.hu
Mobil: 70/259-0425

1. “Hello World !”

Mivel a legtöbb új érdeklődő, windows-os vagy DOS-os rendszereken nőtt fel, először hasonlítsuk a shell scriptet a legtöbbünk által ismert batch fájlokhoz. Ezek a windows-os vagy még inkább a dos-os rendszerekből ismert futtatható fájlok, amikből programokat és dos parancs sorozatokat lehet futtatni. A hasonlat kedvéért tekintsük a dos-t egy shell-nek. Ezeknél a batch fájl voltát, a *.bat kiterjesztés jelölte. Viszont, mivel a linux esetében többféle shell is futhat, ezért fontos, hogy a script-ben meg legyen határozva, milyen shell-re írták. Ezt a fájl első sorában kell jelezni, valahogy így:

```
#!/shell/elérési/utvonala
```

Ha ez nincs benne, akkor a rendszer alap értelmezett shell-jével próbálja azt futtatni. Bár ez a legtöbb mai linux-ban alapból a bash shell. A shell scriptet tartalmazó fájlnál nem kell különleges kiterjesztést használni. Csupán futtathatóvá kell tenni a fájlt.

```
$ chmod +x scriptfajl
```

Közelítsük meg a shell scripteket egy másik hasonlatból. A program kódokat alapvetően két féle képen lehet futtatni. Vagy bináris kóddá fordítja azt egy compiler, vagy futás időben értelmezi egy interpreter. A modern programozásban, lehetőség van mindkettőre. Már a QBasic tervező programjában is, a program írása közben, magát a kódot is le lehet futtatni de ha jól működik, akkor binárissá is fordítható. Az azóta megjelent fejlesztő eszközökben pedig alapvető funkció ez. A shell scriptek esetében sajnos nincs lehetőség binárissá fordítani. A shell scripteket a shell soronként, parancsonként értelmezi és hajtja végre. Úgy is tekinthetünk a scriptekre, mint konzolon, egymás után begépelt parancsokra.

A shell-t héj-programnak is szokták nevezni. Tulajdon képen ez burkolja be a kernelt. A shell a saját nyelvezetében kapott parancsokat a kernel számára értelmezhető módon küldi tovább. Valamint visszafelé, a kernel üzeneteit továbbítja a felhasználónak, vagy a shell-t használó programnak. A shell scriptből, nem csak shell parancsokat adhatunk ki, hanem bármilyen programot futtathatunk is. Sőt, a legtöbb esetben ez történik, ugyanis a "linuxos-parancsoknak" legnagyobb része, önálló program.
Legyen az első példánk, "kötelezően" egy "Hello World !" program. Rajta keresztül a bash-script több jellemzőjét is megvizsgáljuk és egy kis Xdialog-os példát is megnézünk. A parancs sorok előtti "$" jel, a prompt-ot jelképezi, vagyis azt nem kell begépelni.

echo

Nyissunk egy terminál ablakot és írjuk bele a lenti parancsokat ! Természetesen a sorok végén enter-t nyomva.

```
$ clear
$ echo Hello World !
```

Nem nehéz kitalálni mi történt. A "clear" törölte a képernyőt, az "echo" pedig meg jelenítette az utána levő szöveget.
Most tegyük a "Hello World !" szöveget egy változóba és így írassuk ki a képernyőre.
A bash változói, alaphelyzetben szöveges változók. Bár a declare paranccsal lehet integer (egész numerikus érték) típust és tömb típusút is meghatározni. Az integer típusúba, ha szöveges adatot töltünk fel, akkor az nulla értéket ad vissza.

Integer, szám típusú változó létrehozása:
```
$ declare -i $valt
$ valt="szöveg"
$ echo $valt
0
```

Tömb típusú létrehozása:
```
$ declare -a $valt[elemszam]
```

Értékadás:
```
$ valt[elemsorszám]="kifejezés"
```

Érték kinyerése:
```
$ echo ${valt[elemsorszám]}
```

A teljes tömbtartalom kiírása:
```
$ echo ${valt[*]}
```

Az értékkel feltöltött elemek számának lekérdezése:
```
$ echo ${#valt[*]}
```

Ez utóbbi dinamikus tömbként viselkedik, vagyis a deklarációnál meghatározott tömbelemszámnál nagyobb sorszámú elemmel is feltölthető. Viszont a többdimenziós tömbök nem támogatottak.

Minden típus karakteresen tárolódik. A változót alap esetben nem kell külön létrehozni. Mikor értéket kap, akkor létre jön karakteres típusúként. A létrejövő változó, csak arra shell-re érvényes, azaz ha közben becsukjuk a terminál ablakot és újat nyitunk, akkor újra értéket kell adni a változóknak. Nézzük meg a gyakorlatban is, a szöveges típusú változók használatát.
```
$ a=Hello World !
```

A fenti próbálkozás az alábbi hibaüzenetet eredményezi:
```
bash: World: command not found
```

Azt mondja, hogy a "World" parancsot nem találja. Azaz, az első szóköz utáni részt, újabb parancsnak értelmezi.
ennek kivédésére, a szöveget tegyük idézőjelek közé.

```
$ a="Hello World !"
$ echo $a
bash: !": event not found
```

Ez ismét hiba üzenettel tért vissza. Igen, mert a !" kettős karaktert a shell különleges esetekre használja. Tegyünk egy szóközt közéjük.

```
$ a="Hello World ! "
$ echo $a
Hello World !
```

Most oldjuk meg, hogy a szöveg egy részét, dinamikussan változtathassuk. Ezt a szöveg adott részének változóval történő helyettesítésével érhetjüke el.

```
$ b="World"
$ c="City"
$ echo "Hello $b ! "
Hello World !
$ echo "Hello $c ! "
Hello City !
```

Mint láthatjuk, a változó értéke behelyettesítésre került. De mi van, ha ezt nem akarjuk. pl.

```
$ echo "Adj értéket a $b-nak ! "
Adj értéket a World-nak !
```

Mivel a $b változó értéke a "World" szó ezért az echo ezt jeleníti meg a helyén. Ha nem akarjuk, hogy a változó behelyettesítésre kerüljön, akkor használjunk egyes-idézőjeleket.

```
$ echo 'Adj értéket a $b-nak !'
Adj értéket a $b-nak !
```

Ekkor a felkiáltójel utáni szóköz is elhagyható. Az egyes-idézőjelek esetén nem történik semmi különleges dolog végrehajtása. De mivel a legtöbb esetre igen is kell a változó-érték behelyettesítés, leginkább a normál idézőjel használatos. Mit tehetünk akkor, ha egy szövegrészen belül, van olyan szakasz, ahol a változók egy részét szeretnénk, hogy az értékével helyettesítsen a shell és van ahol onkrétan a változó nevét szeretnénk kiíratni.

```
$ szem="József"
$ echo "Kérlek kedves $szem adj értéket a $b változónak ! "
Kérlek kedves József adj értéket a World változónak !

$ echo 'Kérlek kedves $szem adj értéket a $b változónak ! '
Kérlek kedves $szem adj értéket a $b változónak !
```

Egyik esetben sem a kívánt eredményt kapjuk. Két féle megoldás is kínálkozik.

```
$ echo "Kérlek kedves $szem1" 'adj értéket a $b változónak !'
Kérlek kedves Jóska adj értéket a $b változónak !
```

Ekkor két felé vettük az üzenetünket. Egyik felét kettős a másik felét szimpla idézőjelekbe tettük.
De használhatunk egy másik lehetőséget, még pedig azt,hogy 1 db blackslash, "\" jelet teszünk az a változó elé, amelyiket nem szeretnénk az értékével behelyettesíteni. Ez ugyanis az utána következő karakter, a shell számára különleges jelentését hatástalanítja. Úgy is szokták mondani, "levédi" azt.

```
$ echo "Kérlek kedves $szem adj értéket a \$b változónak ! "
Kérlek kedves Jóska adj értéket a $b változónak !
```

Még egy probléma felmerülhet a változók értékének kiíratása során, akkor, ha a kiírandó szövegben a változó után nem akarunk szóközt hagyni. Pl. egy férj nevéhez akarju a "né" jelzőt fűzni:

```
$ nev="Kovács"
$ echo "$nevné"
```

Erre egy üres változót kapunk, mivel a "nevné" nevű változónk üres. A megoldás, hogy a változó nevét jelentő részt és a kiírandó részt a "{}" jelekkel különítjuk el.

```
$ echo "${nev}né"
Kovácsné
```

Bár az eddigi példákat, parancssorba gépelgettük, de ha ezeket egy
`#!/bin/bash`
kezdetű fájlba helyezzük, majd erre futási jogot adunk, akkor ugyanezeket az eredményeket kapjuk.
Tehát:

```
$mcedit hw
```

Ekkor az mc fájlkezelő szövegszerkesztője megnyitja a hw nevű fájlt, vagy ha nem létezik még, akkor mentés során létre hozza. Ebbe begépeljük:

```
----------------------------------------------------------------
#!/bin/bash
a="Hello World ! "
echo $a
```

Maj mentés (F2) után az mcedit-ből kilépve, futási jogot adunk neki:

```
$ chmod +x hw
```

Ha most ki adjuk a
```
$ hw
```
parancsot, azt a hiba üzenetet kapjuk, hogy nincs ilyen program. Ugyan is a $PATH globális változóban lévő útvonalakon keresi. Általában:
/usr/bin:/usr/local/bin:/usr/X11/bin:/home/user/bin
Ha futtatni akarjuk, adjuk meg neki, hogy az aktuális könyvtárba keresse, a "./" karakterekket elé írva:

```
$ ./hw
Hello World !
```

Xdialog példa:

Most megnézzük hogy az Xdialog programot felhasználva, milyen egyszerűen és gyorsan tehetjük script-jeinket látványossá és felhasználóbaráttá.
Ehhez szükséges, hogy az Xdialog program fel legyen telepítve. Az következő oldalról letölthető: http://xdialog.dyns.net/ vagy a http://freshmeat.net/projects/xdialog címről. UHU linux-hoz, a source-t, azaz a forráskódot töltsük le. Könnyedén lefordul. Nincs semmi függősége. Kicsomagolás és a forrásának könyvtárába lépes után:

```
$ ./configure
$ make
$ make install
```

Most az Xdialog legegyszerűbb dobozát fogjuk használni, az msgbox-ot. Ez egy ablak, amiben a "kifejezés" szöveg olvasható és egy "OK" gomb van rajta. A két számmal az ablak mérete adható meg. 0 0 esetén automatikusan a szöveghez méretezi azt. A továbbiakért lásd az Xdialog jegyzetet.

Ha feltelepítettük az Xdialog-ot akkor szintén egy terminálba írjuk be:
```
$ Xdialog --title "Shell Programozás" --msgbox "Hello World ! " 0
0
```

vagy:
```
$ a="Hello World ! "
$ Xdialog --title "Shell Programozás" --msgbox "$a" 0 0
```

Itt fontos az idézőjelek közé tenni a $a változót, mert egyébként nem egységes egészként kezeli az kifejezést, hanem így értelmezi:
```
$ Xdialog --title "Shell Programozás" --msgbox Hello World ! 0 0
```

Ekkor csak a Hello szót írná ki.
Ez érvényes más shell parancsoknál is.

2. Változók és argumentumok

Ebben a részben a változók érték adásának különféle módjairól lesz szó, bele értve a felhasználói adatbevitelt valamint egy parancssorozat eredményének változóba helyezését is. A változó="érték" és a változó='érték' típusú értékadásra itt már nem térnék ki, hiszen azt az előző részben megismertük.

let

Már volt róla szó, hogy a shell változók, alapesetben szöveges változók. Viszont a változókban gyakran numerikus, szám értéket akarunk tárolni. Ezzel nincs is gond, hiszen:

```
$ a="12"
$ echo $a
12
```

Azonban ha az értékekkel, számolni is szeretnénk, nem ilyen egyszerű a dolog.

```
$ b="24"
$ c=$a+$b
$ echo $c
12+24
```

A fenti módszer nem a kívánt eredményt hozta. Mind a változót értékét, mind az operációs jelet, szöveges adatként kezelte. Azt hogy a shell a változó értékét karakteres, vagy numerikus adatként, esetleg dátumként, vagy netán egy fájl neveként értékeli, azt a változó felhasználási módja határozza meg. Tehát a kellő eredmény érdekében meg kell mondanunk a shellnek, hogy hogyan kezelje a változók értékét. Ha azt akarjuk, hogy egy számítást végezzen el, akkor a "let" parancsot kell használnunk.

```
$ let c=$a+$b
$ echo $c
36
```

De ezekben a formában is lehet számolni:
`c=$(($a+$b))`
`c=$[$a+$b]`

A let paranccsal, összeadást, kivonást, szorzást és osztást végezhetünk el és zárójeleket is használhatunk. De csak egész számokat tud kezelni. Azaz a :

```
$ let d=$c/5
$ echo $d
7
```

helytelen eredményt adja.

A tört számokhoz, már külső programot kell használni pl. a "bc" programot.

```
$ echo $c/5 | bc -l
7.20000000000000000000
```

Bár meghatározható integer típusú változó, de ez csak annyit jelent, hogy ha szöveges stringel töltjük fel az így deklarált változót, akkor annak az értéke 0 lesz. Számolni sajnos így sem lehet velük, csak a fent mutatott technikákkal.

```
$ declare -i szam
$ szam="szöveg"
$ echo $szam
0
```

Előfordulhat, hogy egy változó hosszára vagyunk kíváncsiak. Erre megfelelő a wc program használata, amely vissza adja, hogy a bemenetére érkező adat hány sorból, szóból és karakterből áll.

```
$ nev="Kovács Gáspár"
$ hosz=`echo $nev | wc -c`
$ echo "$nev neve $hosz karakterből áll a szóközzel együtt."
Kovács Gáspár neve      14 karakterből áll a szóközzel együtt.
```

read

Igen gyakori az is, hogy a a felhasználótól szeretnénk, adatot bekérni. Erre a shell a "read" utasítást használja. Ekkor egy várakozó, kurzort kapunk. Az adatbevitel során, a kurzormozgató billentyűk is használhatóak. Az adatbevitelt az "enter" leütésével fejezhetjük be. Ekkor a shell folytatja a további utasítások végrehajtását. Feltételezve, hogy a read után pl. az "uhulinux" szót gépeljük be:

```
$read d
$ echo $d
uhulinux
```

Egyszerre több változóba is kérhető be adat. Ekkor a bevitel során a szóközöknél darabolja a begépelt szöveget. Azaz az első szóközig az első változóba, az első és második szóköz közötti részt a második változóba, a második és harmadik szóköz közötti részt a harmadik változóba helyezi, és így tovább. Pl. Tegyük fel, hogy a bevitel során a három adatot, nevet, lakóhelyet és e-mailcímet akarunk bevinni.

```
$read  a b c
$ echo $a
Gábor
$ echo $b
Kecskemét
$ echo $c
nagylevel.nekem@fibermail.hu
```

Felmerül a kérdés, hogy miképpen kezelhető helyesen a szóközöket tartalmazó

kifejezések. Például, ha a névnél a vezeték és a keresztnevet is meg akarjuk adni. Az eredmény ez lesz.

```
$ echo $a
Raffai
$ echo $b
Gábor
$ echo $c
Kecskemét nagylevel.nekem@fibermail.hu
```

Látjuk, hogy amíg van újabb meghatározott változó a read sorban, addig minden szóköz után újba helyezi, a maradékot pedig elhelyezi az utolsóba. Hogyan lehet ezt kivédeni ? Ezt a már ismert backslash "\" karakterrel tehetjük meg. Ez hatástalanítja a mögötte lévő karakter, shell számára történő különleges jelentést. Így a szóközét is. A fenti példánál maradva a következőképpen gépelve a read során, a helyes eredményt kapjuk: "Raffai\ Gábor Kecskemét nagylevel.nekem@fibermail.hu"
A linux parancsok során a szóközökre, és más különleges jelentésű karakterekre oda kell figyelni és a "\" megoldással, valamint az idézőjel tipusok helyes alkalmazásával szabályozni kell, hogy a shell hogyan értelmezzen. A kezdetekben ez egy gyakori és bosszantó hiba forrás lehet.

Az adat bekérése előtt gyakran tájékoztatni kell a felhasználót, milyen adatot is kérünk tőle. Ehhez egy echo parancsot használhatunk előtte.

```
$ echo "Írd be a neved, a várost ahol laksz és az e-mailcímed: "
$ read adatok
$ echo $adatok
Gábor Kecskemét nagylevel.nekem@fibermail.hu
```

Ilyenkor mint láttuk, a kurzor az adatbekéréshez az echo parancs kiírása alatti sorba kerül. Ha azt szeretnénk, hogy a kurzor a tájékoztató szöveg után, vele egy sorba kerüljön, használjuk az echo parancsot "-n" kapcsolóval.

```
$ echo -n "Írd be a neved, a városod, és az e-mailcímed: "
$ read adatok
$ echo $adatok
Gábor Kecskemét nagylevel.nekem@fibermail.hu
```

Előfordul, hogy egy fájl tartamát szeretnénk egy változóba tenni. Vagy legalább is fájlból szeretnénk a változónak értéket adni. Ehhez a "cat" parancsot használhatjuk. Tegyük fel, hogy a fenti adatainkat, az adat.txt fájl tartalmazza.

```
$ cat adat.txt
Gábor Kecskemét nagylevel.nekem@fibermail.hu
```

A cat parancs ebben a formájában a képernyőre írja az adott fájl tartamát. Változóba, a balra dőlő szimpla idézőjelekkel segítségével tehetjük.

```
$ adatok=`cat adat.txt`
$ echo $adatok
Gábor Kecskemét nagylevel.nekem@fibermail.hu
```

A balra dőlő idézőjelek esetén, (amelyet általában a magyar billentyűzeten az AltGr+7 kombinációval érhetünk el,) a két idézőjel közötti részt, mint parancsot végrehajtja a shell és a kapott eredményt helyezi a változóba. Ez akár egy hosszú parancssorozat is lehet.

Az olyan eset is gyakori, hogy egy-egy változó értékét, a script indításakor, argumentum formájában akarjuk megadni. Ezekre a scripten belül, speciális változókkal hivatkozhatunk. Nézzük meg egy példán keresztül, hogy működik ez. Hozzunk létre egy "proba1" nevű fájlt:

```
$ mcedit proba1
```

Töltsük fel az alábbi tartalommal:

```
------------------------------
#!/bin/bash
# 2.fej.1.script
clear
echo "Darab : $#"
echo "Név : $1"
echo "Cím : $2"
echo "E-mail : $3"
------------------------------
```

Mentés után, adjunk neki futási jogot:

```
$ chmod +x proba1
```

Ezután futtassuk három argumentum-all:

```
$ ./proba1 Gábor Kecskemét nagylevel.nekem@fibermail.hu
Darab : 3
Név : Gábor
Cím : Kecskemét
E-mail : nagylevel.nekem@fibermail.hu
```

Mint látjuk a $# különleges változó, a script futtatásakor megadott argumentumok darabszámát tartalmazza.
Megemlítenék még három különleges változót:
$0 A script nevét tartalmazza, pontosabban azt, ahogyan meglett híva.
$* Az összes parancssori argumentumot tartalmazza egyben, egyetlen egységként kezelve.
$@ Az összes parancssori argumentumot tartalmazza, de kölün-külön egységként kezelve.
Ezeken kívül a scriptekből elérhetőek a globális shell változók, mint a $HOME az $USER, az $PATH és a többi.

Kicsit vizsgálódjunk még, a script milyen módon veszi át a parancssori argumentumokat. A scripten kívül létezik nekünk már az $adatok nevű változó, a "Gábor Kecskemét nagylevel.nekem@fibermail.hu" tartalommal, vagy amit Te gépeltél be az előzőek során.
Adjuk át argumentumként ezt a változót a scriptnek, így:

```
$ ./proba1 $adatok
```

Ugyan azt az eredményt kapjuk, mert a script az $adatok változó tartalmát, külön-külön argumentumként kezeli a szóközöknél darabolva. De most a változót tegyük idézőjelbe.

```
$ ./proba1 "$adatok"
Darab : 1
Név : Gábor Kecskemét nagylevel.nekem@fibermail.hu
Cím :
E-mail :
```

Nos ekkor a script, egyetlen argumentumként kezeli a változót tartalmát, mintha a szóközöket "\"-el levédtük volna.

A teljesség kedvéért, próbáljuk még ki, szimpla idézőjellel is.

```
$ ./proba1 '$adatok'
```

A balra dőlő idézőjelet is megnézhetjük:

```
$ ./proba1 `cat adat.txt`
```

Gondolom a fentebbi sorok fényében ezeket már nem kell külön elmagyarázni.

shift

Felmerülhet az igény, hogy az átadásra kerülő argumentumokat sorba feldolgozzuk. Ezt a sift paranccsal tehetjük meg, mely egyel balra lépteti az argumentumok értékét. De nézzük meg egy példán keresztül. proba1 nevű scriptünket, egészítsük ki az alábbi sorokkal.

```
---------------------------------------------
#!/bin/bash
# 2.fej.2.script
clear
echo "Darab : $#"
echo "Név : $1"
echo "Cím : $2"
echo "E-mail : $3"

echo "\$1:$1 \$2:$2 \$3:$3"
shift
echo "\$1:$1 \$2:$2"
shift
echo "\$1:$1"
---------------------------------------------
```

Mint láthatjuk ez sorban kiírja a kapott három argumentumot. Az első shift után az $1-et, azaz a "Gábor"-t eldobja és a helyébe lép a második argumentum, azaz a "Kecskemét". A második helyre, ($2) pedig feljött a harmadik argumentum az e-mailcím. Ekkor már nincs $3.
A második shift után, a jelenlegi $1-et, azaz a "Kecskemét"-et aldobja és a második helyről az e-mailcím feljön a $1-be. A $2 pedig megszűnik.

Xdialog példa:

Befejezésül nézzük meg, miként kérhetünk be adatot egy változóba Xdialog-al.

```
$ adatok=`Xdialog --stdout --title "Shell Programozás" \
--3inputsbox "Írd be a személyes adataidat : " 0 0 \
"Néved: " "" "Városod: " "" "E-mail címed: " ""`
$ echo $adatok
Gábor/Kecskemét Március 15.u./nagylevel.nekem@fibermail.hu
```

Az Xdialog a visszatérési értékeit, még ha az több elemből is áll, mindig egyetlen "sor"-ba adja vissza, egy szeparátorral elválasztva őket. Ez alap esetben a "/". Más is meghatározható a --separator kapcsolóval. Ajánlott a script műveleteivel nem ütköző szeparátort választani. Sajnos szeparátornak csak egyetlen karaktert fogad el.

```
$ adatok=`Xdialog --stdout --title "Shell Programozás" \
--separator ";" --3inputsbox "Írd be a személyes adataidat : " \
0 0 "Néved: " "" "Városod: " "" "E-mail címed: " ""`
$ echo $adatok
Gábor;Kecskemét Március 15.u.;nagylevel.nekem@fibermail.hu
```

Ezután, ha szükséges, az egyes elemeket a cut programmal tehetjük külön-külön változókba, de erről még később lesz szó.

Egy másik érdekes lehetőség az Xdialog inputbox-ainál, a jelszó szerű bevitel lehetősége. A --password kapcsoló használatakor, beviteli mezőbe történő írás során, a bevitt betűk helyén, csak "*" karakterek látszanak. Ha egyszer használjuk a kapcsolót, akkor az utolsó beviteli mezőre érvényes, ha kétszer, akkor az utolsó kettőre, ha háromszor, akkor mindháromra.
Használható az -inputbox, az -2inputsbox és a -3inputsbox esetében. A dobozon megjelenik egy választó kapcsoló amellyel ki és be kapcsolható a "*" effektus, azaz a karakter rejtés.
A részletekért és a jobb megértés végett, lásd a Jegyzetek: Xdialog részt.

3. Szokásos be és kimenet, adatfolyamszerkesztés

Ebben a részben meg ismerkedünk, néhány a windowsos és dos-os világban nem ismert fogalommal, a szokásos bemenet, a szokásos kimenet, a szokásos hiba kimenet és a pipeline, azaz adatfolyam szerkesztés lehetőségével.

Alap helyzetben egy program szokásos bemenete, a billentyűzet, azaz onnan várja a utasításokat. A szokásos kimenet pedig a képernyő, ott jeleníti meg az eredményeket. A szokásos hiba kimenet is a képernyő, a program ide küldi a hibaüzeneteit. Ezeket azért nevezzük "szokásosnak", mert ezek az alapértelmezettek, de ha a célunk úgy kívánja, megváltoztathatóak, átirányíthatóak. Nézzünk 1-2 példát és mindjárt világosabb lesz.

A echo parancsot már ismerjük. Ez egy adott karaktersorozatot, vagy egy változó tartalmát írja ki a képernyőre. Most már fogalmazhatunk pontosabban úgy is, hogy a szokásos kimenetre küldi azt. De mi van, ha a változó tartalmát, nem a képernyőn szeretnénk megjeleníteni, hanem egy fájlba szeretnénk íratni. Ekkor a ">" jellel irányíthatjuk át a szokásos kimenetet, a fájlba. Ha az előző részből még meg van az adat.txt nevű fájlunk, akkor helyezzük a már ismert módon, az "adatok" nevű változóba. (Ehhez természetesen abban a könyvtárban kell állnunk, ahol a fájl található.) Majd a változó értékét, írassuk egy fájlba az echo parancs és a szokásos kimenet átirányításával.

```
$ adatok=`cat adat.txt`
$ echo "$adatok" > uj-adat.txt
```

Ez ha már létezik az uj-adat.txt fájl, akkor felülírja azt, vagyis az előző tartalma elvész és bele írja az $adatok változó értékét. Ha nem ezt szeretnénk, hanem a fájlhoz hozzáfűzni új adatokat, akkor a ">>" jelet kell hasznunk. Hozzunk létre egy "ujadat" nevű változót, töltsük azt fel és fűzessük hozzá a uj-adat.txt fájlhoz, majd ellenőrizzük le meg az eredményt.

```
$ ujadat="András Budapest andras@freemail.hu"
$ echo $ujadat >> uj-adat.txt
$ cat uj-adat.txt
Glindorf Kecskemét nagylevel.nekem@fibermail.hu
András Budapest andras@freemail.hu
```

grep

De nem csak fájlba lehet irányítani egy program kimenetét, hanem egy másik program bemenetére is. Erre a "|" jel szolgál. Ezt hívják adatfolyam szerkesztésnek. Ismerjünk meg itt egy új programot, a grep-et. Ez a bemenetére érkező sorokból, kiszűri azokat, amelyek tartalmazzák a megadott kifejezést és tovább küldi azt a szokásos kimenetén keresztül. De előbb vegyünk fel még egy személy adatait az uj-adat.txt fájlba, ellenőrizzük le sikerült-e, majd a cat programmal irányítsuk a fájl tartalmát a grep program bemenetére, azzal szürjük meg a budapesti lakosokat és az eredményt jelenítsük meg a less programmal.

```
$ echo "Péter Baja peter@mailbox.hu" >> uj-adat.txt
$ cat uj-adat.txt
$ cat uj-adat.txt | grep Budapest | less
```

A cat program kimenetét a grep bemenetére küldtük, majd annak a kimenetét a less program bemenetére. A less pedig az eredményt a szokásos kimenetén, a képernyőn jelenítette meg. A less egy szöveges-fájl nézegető. A megtekintés állapotából, a "q" billentyűvel léphetünk ki belőle (quit).

Egy program bemenetére, a "<" jellel is irányítható egy fájl tartalma. A fenti eredmény a következő módon is elérhető.

```
$ grep Budapest < uj-adat.txt | less
```

Amivel még nem foglalkoztunk, az a szokásos hiba kimenet és annak átirányítása. A programok hibakimenete alaphelyzetben ugyanaz, mint a szokásos kimenet, azaz a képernyő. Tegyük fel, hogy nemlétező fájlt akarunk megnyitni a cat paranccsal.

```
$ cat személyes-adatok
cat: személyes-adatok: Nem létező fájl vagy könyvtár
```

Előfordulhat olyan, hogy a hiba üzeneteket, egy fájlba szeretnénk irányítani. A hiba kimenetre ebben az esetben, a "2>" módon hivatkozhatunk. A normál kimenetre ez alapján így is lehet hivatkozni "1>". De ez azonos a ">" hivatkozással. Úgy is nevezhetjük őket, hogy egyes és kettes kimenet. Vagy normál kimenet és hiba kimenet. A szokásos bemenetet pedig egyszerűen csak bemenetnek. Lássunk egy példát arra, ha a hibaüzeneteket egy "hibák" nevű fájlba szeretnénk gyűjteni. Ekkor a meglévő "hibák" nevű fájl, felülíródik. Ha ezt el akarjuk kerülni és mintegy gyűjteni bele a hibaüzeneteket, akkor a már ismert módon tehetjük meg azt.

```
$ cat személyes-adatok 2>> hibák
```

Ha a hibakimenetet a /dev/null -ba irányítjuk, ezzel nem íródik ki sehová sem.

```
$ cat személyes-adatok 2> /dev/null
```

Lehetőség van arra is, hogy a normál kimenetet és a hiba kimenetet egyszerre kezeljük. Az első példa a normál kimenetet egy fájlba a hiba kimenetet pedig egy másik fájlba irányítja. A második példánál, pedig a hibákat egy fájlba a normál kimenetet pedig egy következő, az esetünkben a less program bemenetére adja tovább.

```
$ cat személyes-adatok > normál 2>> hibák
$ cat személyes-adatok 2>> hibák | less
```

Az is előfordulhat, hogy egy kimenetet fájlba is szeretnénk irányítani és egy másik helyre is átirányítani. Erre a tee program használható. A következő sor az "eredmeny" nevű fájlba írja a normál kimenetet, de a less programmal is megjeleníti.

```
$ cat uj-adat.txt | grep Budapest | tee eredmeny | less
```

Ha a tee utáni részt szabadon hagyjuk, akkor a fájlba is ír és a képernyőn is megjelenít.

```
$ cat uj-adat.txt | grep Budapest | tee eredmeny
```

Lehet két fájlba is íratni a kimenetet és a hibakimenetet pedig előtte egy harmadik fájlhoz hozzáfűzetni. Illetve ezeket lehet kombinálni.

```
$ cat uj-adat.txt  2>> hibafile | tee eredmeny > eredmeny2
```

Néha pedig arra lehet szükségünk, hogy a hiba üzenet is a normál kimenetre menjen. Erre az esetre van lehetőség, hogy az adott kimenetet a másik kimenetre irányítsuk.

Az általunk írt scriptek esetén ugyanígy használhatóak a szokásos ki és bemenetek átirányításai. Csupán két dologgal kell tisztába lennünk. Egyik, hogy az echo parancs mindig a normál kimenetre ír, vagyis, ha a scriptünk futtatásakor annak normál kimenetét átirányítjuk, akkor a scripten belüli összes echo parancs arra a helyre ír. az alábbi esetben a script összes echoja a fájlba íródik.

```
$ sajatscript > fajl
```

A másik pedig, hogy a read parancs ehhez hasonlóan mindig a szokásos bemenetről olvas. Ami alaphelyzetben a billentyűzet. Ezért tudunk vele adatot bevinni. De ha a scriptünk bemenetére irányítunk valamit, akkor a scripten belüli összes read parancs onnan olvas. Ha nincs ott több adat, akkor üres, 0 hosszúságú stringet olvas be. A lenti esetben a script beli read-ok mind a adatfile-t továbbító cat parancs kimenetéről várja az adatot.

```
$ cat adatfile | sajatscript
```

Természetesen ezek a szabályok eléggé megkötik a kezünket egy interaktív script be és kimenetének kezelése terén. Erre is nagyszerű gyógymódot kínál az Xdialog program, ha a dialógust az echo és a read parancsok helyett ezzel vezéreljük le. Ebben az esetben a kimenet-átirányítások szabadon használhatóak.

A scriptünkből a hibakimenetre is tudunk írni. A normál kimenetet "1>", a hibakimenetre "&2" irányítjuk.Az ilyen echo, a hibakimenetre ír.

```
echo "Hiba szöveg" 1>&2
```

Valamint mint minden programnak, a scriptünknek is van egy visszatérési értéke, ami a $? változóban érhető el, közvetlenül az után, hogy a script befejezte a futását. Hogy ennek az értéke mennyi legyen, a scripten belül az exit <érték> vagy exit $x utasítással határozhatunk meg. De ez az érték, csak egy 0 és 255 közötti szám lehet.

Reguláris kifejezések, metakarakterek.

Most ismerkedjünk meg a windowsból bár ismert, de itt mégis másképpen működő dologgal, a shell által ismert reguláris kifejezésekkel. Említhetjük őket metakarakktereknek is, vagy egyszerűen helyettesítő karaktereknek.
Windows-ban és DOS-ban a "*" és a "?" karakterek ismertek. Linuxban több lehetőség is rendelkezésünkre áll.
A "$" jellel a sor elejére,
a "^" jellel pedig a sor végére hivatkozunk.
A "." jellel, egy darab tetszőleges karakterre, (mint dos és windows alatt a "?" jel,)
A "*" jellel pedig a megszokott módon, tetszőleges számú, tetszőleges karakterre hivatkozunk.
A "[" és "]" jellel, a karakterek egy csoportjára tudunk hivatkozni.
[0-9] : Bármilyen szám karaktert jelent.
[A-Z] : Bármilyen nagybetűs karaktert jelent.
[skZ4o] : Bármelyik karaktert jelenti a felsoroltak közül.
Érdemes még meg említenünk, hogy kifejezéseken belül a TAB-ra a "\t" a sorvégre pedig a "\n" módon hivatkozhatunk

Fontos megjegyezni, hogy egykét program egy-egy reguláris kifejezést másképpen használ, vagy másképpen jelöl. Ezért ha valami nem jól működik, akkor nézzük át az adott program man-ját. Ilyen pl. a grep program "*" metakarakter értelmezése, mely jelentése a shellben, "bármennyi számú, bármilyen karakter". A grep-ben pedig, "bármennyi számú, a "*" jel elött álló karakter". Azaz a "a*" jelentése a shell-ben: egy "a" karakterel kezdődő bármilyen string. Mig ugyanennek a jelentése grep-ben: Az "a" karakter bármennyiszer-i ismétlődése. Pl. "a" "aaa" vagy "aaaaaaa". Mivel a grep a "." meatakaraktert ugyanúgy értelmezi mint a shell, azaz egy darab bármilyen karakter, ezért a shell beli "*" reguláris kifejezés, grep-es megfelelője a ".*", azaz a "bármilyen karakter bármennyiszer-i ismétlődése".

A kód formázása

Mivel rövidesen már valódi scripteket fogunk írni, ezért ismerkedjünk meg azokkal a lehetőségekkel, amikkel olvashatóbbá és át tekinthetőbbé tehetőek a scriptjeink.
Az egyik alapvető dolog, hogy a logikailag összetartozó részeket megfelelő számú üres sorral választjuk el egymástól. Ez a függőleges tagolás. A másik, a vízszintes tagolás, ahol az program végrehajtás soráni más-más mélységben lévő sorokat, bentebb kezdjük mint az előzőt. Ezzel az elágazások és ciklusok tehetők át tekinthetőbbé.

A harmadik dolog a kommentek használata. A sorban a "#" jel utáni rész, már nem hajtódik végre. Ha ez az első karakter, akkor értelem szerűen az egész sor kimarad a végrehajtásból.

```
read a; echo $a   # a további szövegrész nem hajtódik végre.
# read a; echo "$a" ez a sor egyáltalán nem hajtódik végre.
```

Már ismerjük a "\" jelet és hatását. Ezzel a sor végén lévő sorvégjel is hatástalanítható, vagyis ha a sor végén egy "\" szerepel, akkor a shell úgy veszi, hogy nem történt uj sor kezdése, vagyis a következő sort is az előző folytatásaként hajtja végre. Ebben az esetben fontos, hogy a "\" jel után már nem állhatnak karakterek, még szóközök sem, illetve abból egy igen. Ebből ered, hogy ha az ilyen eltört sorokban kommenteket akarunk elhelyezni, akkor a "\" és a "#" között, maximum egy space lehet.
A sorvégjel „\”-el történő levédése, akkor is hasznos lehet, ha olyan hosszú parancssorokat használunk, amik esetleg nem férnek ki egy sorba. (Már mint nem a terminálba, mert ott ez nem okoz gondot, hanem a dokumentumba, ahol a példák szerepelnek.)
Az alábbi parancssor,

```
$ cat file.txt | grep Budapest | cut -d\; -f 2-4 | tr '[A-Z]'
'[a-z]' > ujfile.txt
```

alábbi módon történő leírása ugyan azt jelenti.

```
$ cat file.txt |\
 grep Budapest |\
 cut -d\; -f 2-4 |\
 tr '[A-Z]' '[a-z]'\
 > ujfile.txt
```

A megtört parancssort a kódban TAB-okkal is bentebb helyezhetjük, a futást ez nem zavarja. De ha egy ilyen szakaszt kimásolunk a dokumentumból a vágólapon keresztül, egy terminál parancssorába, ott hibásan fog lefutni.

```
    $ cat file.txt | grep Budapest | \
    cut -d\; -f 2-4 | tr '[A-Z]' '[a-z]' \
    > ujfile.txt
```

A parancssor-törés ellenkezője is megoldható, azaz több parancsot is írhatunk eggyetlen sorba. Több parancs is írható egyetlen sorba, a ";" jellel elválasztva.

```
$ echo -n "Mi a neve ?"
$ read nev
$ echo "Üdvözlöm $nev."
```

Ezt írhatjuk egyetlen sorba is.

```
$ echo -n "Mi a neve ?"; read nev; echo "Üdvözlöm $nev."
```

4. Adatfolyam szűrők és szerkesztők

Ebbe a részben néhány egyszerűbb szöveg és szöveges fájlt manipuláló lehetőséget ismerünk meg. Ezekre mint szűrőkre is szoktak hivatkozni. Mivel a programok egyébként csak a legszükségesebb mértékben vannak ismertetve, érdemes az élesben történő használatuk előtt a man-t elolvasni, bár ez nem feltétele annak, hogy az itt leírtak megérthetőek legyenek.

Hozzunk létre egy fájlt az alábbi adatsorokat beleírva, adatok.dat néven. Az egyes adatsorokat, írjuk egyetlen sorba és ne hagyjunk közöttük üres sorokat.

```
Sorszám;Név;Születési hely;Születési idő;Anyja neve;Gyermekkori
leányneve;Lakhely;Nem;Családi állapot;Gyermekek száma

1;Kovács Gábor;Budapest;1973 Január 14;Mardin
Éva;;Budapest;férfi;házas;2

2;Bárdos Péter;Szombathely;1965 Március 24;Kele
Katalin;;Pécs;férfi;független;0

3;Szeghalminé Éva;Budapest;1957 Február 05;Péteri Eszter;Konkoly
Éva;Cegléd;nő;házas;1

4;Almosné Szabó Renáta;Kecskemét;1976 December 20;Ostoros
Sára;Szabó Renáta;Budapest;nő;független;1

5;Izsó Péter;Zánka;1953 November 09;Kővári
Piroska;;Budapest;férfi;elvált;3
```

Itt csak a legegyszerűbb szűrési és karakter cserélési lehetőségekkel fogunk megismerkedni.

grep , cut

A grep programot már ismerjük. Ez is egy szűrő, amely a bemenetére érkező sorok közül csak azokat küldi tovább, amelyek a megadott feltételnek megfelel. Listázzuk ki a "Budapest" szót tartalmazó sorokat.

```
$ cat adatok.dat | grep Budapest
1;Kovács Gábor;1973 Január 14;Mardin Éva;;Budapest;férfi;házas;2
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly
Éva;Cegléd;nő;házas;1
4;Almosné Szabó Renáta;1976 December 20;Ostoros Sára;Szabó
Renáta;Budapest;nő;független;1
5;Izsó Péter;1953 Nő
```

A grep -v kapcsolójával, megfordíthatjuk a feltételt, azaz azok a sorok listázódnak, amelyek nem felelnek meg a feltételnek.

```
$ cat adatok.dat | grep -v Budapest
0;Név;Születési idő;Anyja neve;Gyermekkori
leányneve;Lakhely;Nem;Családi állapot;Gyermekek száma
2;Bárdos Péter;1965 Március 24;Kele
Katalin;;Pécs;férfi;független;0
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly
Éva;Cegléd;nő;házas;1
```

A grep alaphelyzetben mint a linux is, kis és nagybetű érzékeny. Ezt a grep-nél az -i kapcsolóval felfüggeszthetjük.

```
$ cat adatok.dat | grep péter
```

Így nincs találat.

```
$ cat adatok.dat | grep -i péter
2;Bárdos Péter;1965 Március 24;Kele
Katalin;;Pécs;férfi;független;0
5;Izsó Péter;1953 November 09;Kővári
Piroska;;Budapest;férfi;elvált;3
```

Így viszont azonosnak veszi a kis és a nagy betűket.

A -w kapcsolóval, csak a teljes szavakra történő illeszkedést veszi megfelelésnek. Az -x kapcsolóval lehetőség van egész sort is megfeleltetni.

Tegyük fel, hogy az egy gyermekes nők nevére van szükségünk. Ekkor az eredményt tovább kell küldenünk több szűrésen is.

```
$ cat adatok.dat | grep ';nő;' | grep 1
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly
Éva;Cegléd;nő;házas;1
4;Almosné Szabó Renáta;1976 December 20;Ostoros Sára;Szabó
Renáta;Budapest;nő;független;1
```

Az adatokat, akármennyi szűrésen átküldhetjük. De mint említettük, nekünk csak az érintettek nevére van szükségünk. Erre a cut programot fogjuk használni. Ebben az adat fájlban, az adatmezők szeparátoraként a ";" karakter szerepel. Ezt közölnünk kell a cut-al a -d kapcsolójával. Ezután meg kell adnunk, hogy az adatmezők közül, hányadikra van szükségünk. Ezt az -f kapcsolójával tehetjük meg. Ne felejtsük levédeni a ";" karaktert, mert különben nem tudja majd a shell értelmezni a parancssort, ugyan is az ez a jel után következő részt, újabb parancssornak értelmezi.

```
$ cat adatok.dat | grep ';nő;' | grep "\;1$" | cut -d\; -f2
Szeghalminé Éva
Almosné Szabó Renáta
```

Tételezzük fel, csak az egyik egyén adataira lenne szükség. Hogyan lehetne egyszerűen megcsinálni, hogy itt választani lehessen a felsoroltak közül és annak az egynek megjeleníteni az összes adatát ? Ehhez ne csak a nevet, hanem az első adatoszlopot, az index mezőt, ami a mi egyszerű esetünkben egyben az adatsor sorszámát is jelöli. Ezután kérjünk egy választást a felhasználótól és annak alapján az adott személy összes adatát jelenítsük meg. Ez nem is olyan bonyolult, mint gondolnánk.

```
$ cat adatok.dat | grep ';nő;' | grep "\;1$" | cut -d\; -f 1-2
3;Szeghalminé Éva
4;Almosné Szabó Renáta

$ echo -n "Hányas sorszámú személy adatai kellenek ?"
$ read v
$ cat adatok.dat | grep "^$v\;"
3;Szeghalminé Éva;1957 Február 05;Péteri Eszter;Konkoly
Éva;Cegléd;nő;házas;1
```

Amennyiben a bement nem egy, hanem több sorból áll, akkor minden sor adott mezője tovább adódik. Így egy adatfájlból egész mezők vihetők át egy másik fájlba. Tegyünk is egy próbát, bár ne küldjük az eredményt fájlba.

```
$ cat adatok.dat | cut -d\; -f 2-4
Név;Születési idő;Anyja neve
Kovács Gábor;1973 Január 14;Mardin Éva
Bárdos Péter;1965 Március 24;Kele Katalin
Szeghalminé Éva;1957 Február 05;Péteri Eszter
Almosné Szabó Renáta;1976 December 20;Ostoros Sára
Izsó Péter;1953 November 09;Kővári Piroska
```

Nem csak egy mezőt, hanem több mezőt is lekérhetünk. Az "-f 1,3,4" vagy az "-f 2-4" módon. Az előbbi esetben az 1-es, 3-as és 4-es mezők, az utóbbi esetben pedig a 2-től a 4. mezők kerülnek tovább adásra.

A cut programmal nem csak adatmezőket választhatunk ki, hanem megadott pozíciójú karaktereket is. Bár ennek ebben a példában nincs értelme, de ha az eredményből, tegyük fel, csak a 6-8-dik karakterekre lenne szükségünk, ez is megoldható.

```
$ cat adatok.dat | grep ';nő;' | grep \;1$ | \
cut -d\; -f2 | cut -c 3-7
eghal
mosné
```

Mivel két soros volt az utolsó cut bemenete, ezért mindkét soron végrehajtotta. A több soros bemenet szétbontása is megoldható később bemutatandó technikákkal.

A "-c 2,4,7" módszerrel nem egymás melletti karakterek is megadhatók.

sort, join

Ezeket az adat fájlokat időnként érdemes karbantartani. Esetleg egy másik adatbázisfájlt szeretnénk már egy meglévőből készíteni, de más elrendezéssel. Ezekhez alkalmas lehetőségeket fogunk most megvizsgálni.

Egy szöveges adatbázis fájl rendezéséhez a sort parancs használható. Tegyük fel, hogy a adatok.dat fájlunkat szeretnénk rendezni a nevek szerint abc sorrendbe. Ezt az alábbi módon tehetjük meg.

```
$ cat adatok.dat | sort -t ';' +1 > adatok-rendezett.dat
```

A -t kapcsoló után kell megadni a fájlban a mezőhatároló karaktert. a "+" után van megadva, hogy hányadik mező szerint rendezzen. Az első mező, a nulladik mező. Vagyis, ha a 3. mező szerint akarunk rendezni, akkor "+2"-t kell írni. Ezt a -k kapcsolóval lehet elkerülni. (Ez az első adatsort is, ami a mező neveket tartalmazza, belekeveri a valódi adatsorok közé. A probléma orvosolható, de most nem térünk ki rá.)

További lehetőségek a finomításra következő kapcsolók:
-k szám Ha ezzel, nem pedig a +szám kapcsolóval hivatkozunk a mezőre, akkor az első mező sorszáma nem 0, hanem 1.
-d Rendezésnél figyelmen kívül hagyja a különleges karaktereket és írás jeleket.
-f Különbséget tesz a kis és nagybetűk között. (Tehát alaphelyzetben nem tesz különbséget.)
-n Numerikus adatok szerint rendez. Ez a többjegyű számoknál lényeges.
-M Az első három karaktert, hónapok neveiként értelmezi és rendezi.
-r Fordított rendezési sorrendet eredményez.
-u Törli a teljesen megegyező sorokat.

Láttuk, miként lehet a cut programmal. egy adatfájlból, teljes mezőket kinyerni, vagy másik fájlba helyezni. Most nézzük, miképpen lehet két adatfájlt egyesíteni. Erre a join parancs alkalmas. Fontos, hogy a két adatbázisnak azonos kulcs szerint rendezettnek kell lennie. Elötte mindkét adatbázist az egyik mező szerint rendezzük. Ha az egyik adatbázis a neveket és címeket tartalmaz, a másik pedig neveket és telefonszámokat, akkor rendezzük mindkettőt a neveket tartalmazó mező szerint. Az egyesítés során, a mindkét fájl álltal tartalmazott neveknél, a sor már három mezőből fog állni. név, cím, telefonszám.

```
$ join -t ';' nev-cím.txt nev-tel.txt > egyesült.txt
```

A -t kapcsolóval a mezőhatároló karaktert adjuk meg. Ha nem az első mező alapján történt a rendezés és történik az egyesítés, azt a -j kapcsolónál adhatjuk meg. A -j kapcsoló után közvetlenül 1-es vagy 2-es szám kell szerepeljen. Ebből a kívánt működés érdekében 1-est írjunk. Ez után kell megadni, hányadik mező szerint történjen az egyesítés. Feltételezve, hogy a fenti két fájl három mezőből áll és a második a név mező akkor így kell végrehajtani.

```
$ join -t ';' -j1 2 nev-cím.txt nev-tel.txt > egyesült.txt
```

tr, sed

Most nézzünk olyan szűröket, amikkel karakterek cserélhetők fel, vagy távolíthatók el.
A legegyszerűbb a tr program. Előbb használati módját, majd néhány egyszerű példát mutatok.

```
$ b=`echo "$a" | tr "cdk" "rht"`
```

Ez a sor, az $a változó értékében, a "c" karaktert "r"-re, a "d"-t "h"-ra, a "k"-t pedig "t"-re cseréli, majd az új karaktersort a $b változóba írja.

A -d kapcsolóval törli a szövegből a megadott, ebben az esetben a "c" és "d" karaktereket.

```
$ b=`echo "$a" | tr -d "cd"`
```

Az alábbi módon a nagybetűket lehet kisbetűkre cserélni.

```
$ b=`echo $a | tr '[:upper:]' '[:lower:]'`
```

Így pedig egy fájlban lehet a tab és sorvég karaktereket átalakítani.
Ez az egyik fájlt a másikba másolja, de közben a TAB-okat ";" karakterre cseréli. (Általában a szöveges adatbázisok, az adatmezők elválasztására, e kettő közül valamelyiket szokták használni.)

```
$ cat fajl | tr '\t' ';' > masikfajl
```

Így pedig a sorvég jelek távolíthatók el egy fájlból.

```
$ cat fajl | tr -d '\n' > masikfajl
```

Ez pedig a ";" mezőhatároló karaktereket cseréli sorvégjelre, így a mezőket külön-külön sorba helyezi. Később szükségünk lessz erre a megoldásra.

```
$ cat fajl | tr ';' '\n' > masikfajl
```

Az -s kapcsolóval az ismétlődések távolíthatók el. Az alábbi sor a felesleges space-ket távolítja el

```
$ echo "Kovács    Béla         Budapest." | tr -s ' '
Kovács Béla Budapest.
```

De ugyan ezt az eredményt adja a tr -s ' [:space:]' is.

Az -s kapcsoló, az egymás után következő, az argumentumban megadott típusú karaktert von össze, egyetlen egybe.

A tr 1-1 karaktert tud kezelni. Amennyiben több karakterből álló karakter sorozatot szeretnénk egy másik, akár a lecserélendővel nem is azonos hosszúságú karaktersorozatra cserélni, ehhez a sed program használható. Ehhez így használjuk. A reguláris kifejezések természetesen a sed-nél is használhatók. A sed egy komoly program, rengeteg lehetőséggel. Külön könyvet is lehetne róla írni, ezért én most nem is mennék nagyon bele, csupán néhány példát mutatok.

Ez a bemenetére érkező adatsorban, az össze kif1 karaktersort, a kif2 karaktersorra cseréli.
```
sed s/kif1/kif2/g
```

Csak a sor elején szereplő kifejezéseket cseréli ki.
```
sed s/^kif1/kif2/g
```

Csak a sorvégi kifejezéseket cseréli ki. A „$” jel mint reguláris kifejezés része, mint metakarakter, nem tévesztendő össze a változók elején szereplő jelöléssel.
```
sed s/kif1$/kif2/g
```

Ez pedig minden sor elejére az adott kifejezést szúrja be.
```
sed s/"^"/"kif"/g
```

Hogyan lehet egy fájl, vagy egy bemeneten érkező adatfolyam kívánt sorszámú sorát megkapni.
```
sed -n "$sorszam p"`
```

Természetesen az adatbázis kezelés sokkal összetettebb problémákat vet fel és a megoldásukhoz is összetettebb megoldások szükségesek. Az itt bemutatott példák csak azt a célt szolgálják, hogy az alapvető lehetőségek bemutatásra kerüljenek.

Xdialog példa:

Illő módon fejezzük be ezt a részt is egy Xdialog-os példával.
Erre válasszuk a timebox-ot, mely három spin-t, azaz olyan kis beviteli lehetőséget tartalmaz, amiben numerikus értékek állíthatók be. Kezdeti értéke az aktuális idő, óra, perc és másodperc értéke. Nyugodtan állítgathatjuk, ez nem változtatja meg a gépünk időbeállítását. Arra külön kellene a kódban root jogosultsággal utasítani. A doboz visszatérési értéke a beállított idő, ":"-al elválasztva az óra a perc és a másodperc. Ezt cut-al külön-külön változókba tesszük és egy infobox-al kiíratjuk, amit 5 másodpercre jelenítünk meg és letiltjuk az OK nyomógomb megjelenését, valamint az ablak bezárhatóságát is. A következő kódot helyezzük egy futási joggal rendelkező, #!/bin/bash kezdetű fájlba.

```
-------------------------------------------------------------------------
#!/bin/bash
# 4. fejezet 1. script

ido=`Xdialog --stdout --title "Shell Programozás" \
--timebox "Ön szerint mennyi a pontos idő ?" 0 0`

ora=`echo $ido | cut -d: -f 1`
perc=`echo $ido | cut -d: -f 2`
mp=`echo $ido | cut -d: -f 3`

Xdialog --no-buttons --no-close \
--title "Shell Programozás " \
--infobox "Ön szerint, $ora óra,\n$perc perc, \
és $mp másodperc van ." 0 0 5000
-------------------------------------------------------------------------
```

A timebox esetében az --stdout biztosítja, hogy az eredménye ne a hiba kimenetre, hanem a normál kimenetre menjen. Az infobox esetében pedig a --no-buttons kapcsoló letiltja a gombok megjelenését, a --no-close pedig lehetetlenné teszi az ablak bezárását. Az infobox álltal megjelenített szövegben a "\n" karakterek utasítják sortörésre a szöveg megjelenítésekor. A legvégén található, harmadik szám pedig azt határozza meg, mennyi idő múlva záródjon be magától az ablak, ezredmásodpercben kifejezve.

5. Feltétel vizsgálat, feltételes vezérlési szerkezetek

Ebben a részben az egész scriptünk struktúráját és működét leginkább maghatározó technikákkal, az elágazásokkal és ciklusokkal ismerkedünk meg. Ezek, nagy része, különféle feltételvizsgálatokra épül.

test

Egy feltétel vizsgálatára hogy az igaz-e, a test parancsot használjuk. Ezt nem önmagában, hanem egy elágazás vagy ciklus szerves részeként használjuk. Néhány egyszerűbb vizsgálat következik.

Fájlok, könyvtárak vizsgálata.
test -d file
Igaz ha a file létezik és könyvtár.
test -e file
Igaz ha a file létezik.
test -f file
Igaz ha a file létezik és szabályos fájl.
test -r file
Igaz ha a file létezik és olvasható.
test -w file
Igaz ha a file létezik és írható.
test -x file
Igaz ha a file létezik és végrehajtható.

Szöveges kifejezések vizsgálata.
test -z string
Igaz ha a string 0 hosszúságú.
test -n string
string Igaz ha a string nem 0 hosszúságú.
test string1 = string2
Igaz ha a stringek megegyeznek.
test string1 != string2
Igaz ha a stringek nem egyeznek meg.

Numerikus értékek vizsgálata.
test numkif1 OP numkif2
A következő operátorokat (OP) ismeri.
-eq egyenlő
-ne nem egyenlő
-lt kisebb mint
-le kisebb vagy egyenlő
-gt nagyobb mint
-ge nagyobb, vagy egyenlő
A numerikus kifejezés negatív is lehet, de egész számnak kell lennie.

A test ismer egy speciális numerikus kifejezést, mely a string hosszát jelenti.

-l szöveg

Egy 5 karakteres szövegrész esetén az alábbi vizsgálat igazként értékelődik ki.

test 5 -eq -l kif

A vizsgált feltétel eredményét negálja az elé írt "!" jel. Azaz, ha igaz, akkor hamissá, ha pedig hamis, akkor igazzá fordítja.
Tehát hamis lesz a

```
test ! 5 -eq 5
```

és igaz a

```
test ! 2 -eq 5
```

Több feltétel vizsgálata is összekapcsolható az -a (and) "és" illetve a -o (or) "vagy" logikai operátorokkal.

```
test $nev = "Gábor" -a $kor -eq 30
```

Ez akkor bizonyul igaznak, ha a $nev értéke "Gábor" és a $kor értéke pedig 30

```
test $kor -le 17 -o $kor -ge 61
```

Ez akkor igaz, ha a $kor értéke, a 18 és 60 közötti intervallumon kívül esik.

If; then ;else; fi

Az elágazásokat, feltételes vezérlési szerkezetként is szokták említeni. Ebből kettőt vizsgálunk meg.

Az if elágazás szerkezete a következőképpen néz ki.

if test <feltétel>
then
    <utasítások>
else
    <utasítások>
fi

A then és az else közötti utasítások akkor hajtódnak végre, ha iga a kifejezés, az else utániak pedig akkor, ha hamis.

Az if elágazások egymásba is ágyazhatók.

```
if test <feltétel1>
then
    <utasítások1>

     if test <feltétel2>
     then
          <utasítások2>
     else
          <utasítások3>
     fi

else

    <utasítások4>

            if test <feltétel3>
     then
          <utasítások5>
     else
          <utasítások6>
     fi

fi
```

Ez a szerkezet, ha a feltétel1 igaz, akkor végrehajtja az utasítások1-et, majd megvizsgálja a feltétel2-őt és ha az is igaz, akkor végrehajtja az utasítások2-őt, ha nem igaz akkor az utasítás3-at. Ha a feltétel1 nem igaz, akkor végre hajtja az else utáni részt, azaz az utasítások4-et, majd megvizsgálja a feltétel3-at, ha az igaz, akkor végrehajtja az utasítások5-öt, ha nem igaz akkor az utasítások6-ot.
Az utasítássor TAB-okkal történő rendezése, nem véletlen a könnyebb átláthatóságot szolgálja. A struktúrálisan egy szintbe, mélységbe tartozó szakaszok előtt, azonos számú TAB van.

Vizsgáljunk meg néhány konkrét példát is. Hozzunk létre egy #!/bin/bash kezdetű fájlt és adjunk neki futási jogot. A további példákat ebbe írjuk és az eredményt a script futásakor tekinthetjük meg. Töltsük fel az alábbi, két vonal közötti tartalommal.

Itt már a magyarázatok egy részét kommentekben adom meg.

```
-------------------------------------------------
#!/bin/bash
# 5.fej. 1.script

clear   # képernyő törlés

# Megszámolom, hány adatsor van az adatbázisomban,
# törlöm az eredményből a számolást bezavaró space-ket,
# kivonok belőle egyet, mert az első, a 0-dik,
# ami a mezők neveit tartalmazza.
let db=`cat adatok.dat | wc -l | tr -d ' ' `-1

# Bekérem, hanyas sorszámú személy legyen feldolgozva.
echo -n "Adja meg a feldolgozandó személy \
sorszámát (1-$db) : "; read a

# Le ellenőrizzük megfelelő-e a sorszám amit a felhasználó adott.
# test akkor igaz, ha a felhasználó által adott szám,
# kisebb 1-nél, vagy nagyobb az adatbázisban lévő
# személyek számánál.A then és a fi közötti rész
# csak ekkor hajtódik végre.
if test $a -lt 1 -o $a -gt $db
then
    # A script tájékoztatja, a felhasználót,
    # hogy rossz számot adott meg,
    # vár egy enter leütéséig, majd kilép a scriptből.
    echo "Ön érvénytelen adatot adott meg ! "
    echo "A script a futását befejezi. Kérem nyomjon entert."
    read x   # az enter leütéséig felfüggeszti
             #a script futását
    exit     # azonnal kilép a scriptből
fi

# A megadott sorszámú személy teljes adatsorának bekérése.
# "^$a\;" jelentése: ha a sor elején található az $a értéke,
# utána pedig egy ";" jel következik.
# Azaz ha az első mező értéke azonos $a-val.
szem=`cat adatok.dat | grep "^$a\;"`

# Adatmezőinek külön-külön változóba helyezése.
nev=`echo $szem | cut -d\; -f2`
szulido=`echo $szem | cut -d\; -f3`
anyneve=`echo $szem | cut -d\; -f4`
leanyneve=`echo $szem | cut -d\; -f5`
lakhely=`echo $szem | cut -d\; -f6`
nem=`echo $szem | cut -d\; -f7`
csalall=`echo $szem | cut -d\; -f8`
gyermsz=`echo $szem | cut -d\; -f9`
```

```
# a date megadja a mai dátumot és időt,
# a cut-al kinyerjük belőle az évszámot.
datum=`date | cut -d'.' -f1`

# a $szulido-bol kivesszük a születési évet.
szulev=`echo $szulido | cut -d' ' -f1`
let kor=$datum-$szulido   # kiszámoljuk az alany életkorát.

# Meg vizsgáljuk, az alany házas-e,
# ha igen akkor a then és az else,
# ha nem, akkor az else és a fi közötti rész hajtódik végre.
if test $csalall="házas"
then
        # Ez mindenképpen kiíródik:
        echo -n "$nev, $kor éves, \
        $lakhely-i lakos, házasságban élő, "

        # Ha a gyermekek száma 0 akkor a then,
        # egyébként az else utáni rész hajtódik végre.
        if test $gyermsz -eq 0
        then
            echo "gyermektelen $nem."
        else
            echo "$gyermsz gyermekes $nem."
        fi

else
    # Ez mindenképpen kiíródik:
    echo -n "$nev, $kor éves, $lakhely-i $nem."

    # Ha a gyermekek száma 0 akkor a then,
    # egyébként az else utáni rész hajtódik végre.
      if test $gyemek -eq 0
      then
          echo "Családi állapota $csalall, nincs gyermeke."
      else
          echo "Családi állapota $csalall, $gyersz gyermeke van."
      fi

fi
-----------------------------------------------------
```

Újdonságok:

clear
  Töröli a képernyőt.

```
tr -d ' '
```
  A tr -d kapcsolója esetén elég csak egy kifejezést meg adni, mert az abban megadott részeket törli. Az esetünkben a szóközöket.

`read x`
Felfüggesztjük a script futását, amíg a felhasználó kényelmesen el nem olvassa az információt, majd le nem nyomja az enter billentyűt. A read x -et használjuk, vagyis kérjük egy változó feltöltését a felhasználótól, de ennek nincs célja, csak az, hogy az enter megnyomásáig felfüggesszük a futást.

`exit`: Hatására azonnal befejezi a futását a script.

Case; esac

Most egy újabb elágazás típussal fogunk meg emlékezni a case-el.
A szerkezete a következő.

```
case $változó in
  érték1)
        <utasítások1>
     ;;
  érték2)
        <utasítások2>
     ;;
  érték3)
        <utasítások3>
     ;;
  érték4)
        <utasítások4>
     ;;
  *)
        <utasítások5>
     ;;
esac
```

Bár a fenti feltételes végrehajtások, if elágazásokkal is megoldhatók lennének, mint látható, ez sokkal áttekinthetőbb és elegánsabb. Fontos hogy az utasításokat két ";" jellel, azaz ";;" módon zárni. Ez egy break-et vált ki, azaz a további vizsgálatokon nem megy végig a program, hanem kilép a case-ből. Ez gyorsítja a vizsgálatot. Ha csak nem éppen az a célunk, hogy a további eset is ki értékelődjön. Az utolsó, *) meghatározás utáni <utasítások5> akkor hajtódik végre, ha egyik előtte lévő feltétel sem teljesült. Tulajdonképpen hasonlít a if else ágához.
A case segítségével, könnyen létrehozhatunk menü szerű működést is a scriptben. Nézzük meg a case működését egy ilyen példán keresztül. Hozzunk létre egy fájlt, másoljuk az két szaggatott vonal közötti részt bele, majd adjunk rá futási jogot.

```
-----------------------------------------------------
#!/bin/bash
# 5.fej. 2.script

clear

# Kiírom a főmenüt.
echo
echo "Kérem válasszon a menüpontok előtti szám bevitelével."
echo "---------------------"
echo "1.Fájl"
echo "2.Szerkesztés"
echo "3.Eszközök"
echo "4.Beállítások"
echo "5.Kilépés"

read v   # Bekérem az első választást.

# Az első válasz vizsgálata:
case $v in   # a külső case eleje. (főmenü)
1)
   # Amennyiben a főmenü 1. pontját választotta.
   clear
   # Kiírom, az első almenüt.
   echo "Kérem válasszon a menüpontok elötti szám bevitelével."
   echo "---------------------"
   echo "1.Megnyitás"
   echo "2.Mentés"
   read v   # Bekérem a második választást.

    # Az első almenü válaszának vizsgálata.
    case $v in   # Az első, belső case. (1.almenü)
    1)
       # Amennyiben a főmenü 1. pontjának,
       # 1. almenüjét választotta.
       echo "Ön egy már meglévő dokumentumot akar kinyitni."
    ;;
    2)
       # Amennyiben a főmenü 1. pontjának,
       # 2. almenüjét választotta.
       echo "Ön menteni szeretné a dokumentumot."
    ;;
    esac
;;
```

```
2)
   # Amennyiben a főmenü 2. pontját választotta.
   clear
   # Kiírom, a második almenüt.
   echo "Kérem válasszon a menüpontok elötti szám bevitelével."
   echo "--------------------"
   echo "1.Másolás"
   echo "2.Kivágás"
   echo "3.Beillesztés"
   read v   # Bekérem a második választást.

    # Az második almenü válaszának vizsgálata.
    case $v in   # A második, belső case. (2.almenü)
    1)
       # Amennyiben a főmenü 2. pontjának,
       #1. almenüjét választotta.
       echo "Ön kivágott egy részt."
    ;;
    2)
       # Amennyiben a főmenü 2. pontjának,
       # 2. almenüjét választotta.
       echo "Ön kimásolt egy részt."
    ;;
    3)
       # Amennyiben a főmenü 2. pontjának,
       #3. almenüjét választotta.
       echo "Ön egy részt illesztett be."
    ;;
    esac
;;

3)
     # Amennyiben a főmenü 3. pontját választotta.
     echo "Ön az eszközöket választotta."
;;
4)
     # Amennyiben a főmenü 4. pontját választotta.
     echo "Ön a beállításokat választotta."
;;
5)
     # Amennyiben a főmenü 5. pontját választotta.
     echo "Most kilépek."
      sleep 3
      exit
;;

esac   # a külső case vége. (főmenü)
-----------------------------------------------------
```

A példából hiányzik, a felhasználói válaszok helyességének vizsgálata, de most csak a case megismerése a cél. Bár a válaszok, csak egy karakteresek, azért a case elágazások jól követhetők. Az első két case elágazásnál, további menüpontokat adtunk meg és az azokra történő válasz alapján egy újabb belső case vizsgálat indult, mindkét esetben. Ezzel a módszerrel, illetve ennek a finomításával egyszerű menüvel láthatjuk el a scriptünket.

Nézzük meg az Xdialog-al, hogyan lehet menüt csinálni. Ugyanezt a fenti menüt készítsük el. Az előzőekhez képest csak annyi a különbség, hogy echo és read helyett használjuk a dialógus-ablakokat. Az Xdialog jegyzetből nézzük át a --yesno a --fselect ás a --menubox dialógus-ablakok leírását.

```
-------------------------------------------------
#!/bin/bash
# 5.fej. 3.script

clear

until test
do

valasz=`Xdialog --stdout --title "Shell Programozás 5/3 script" \
--backtitle "Menü Próba" --no-cancel --no-tags \
--item-help --menubox "Válasszon a menüből." 0 0 6 \
"1" "Fájl" "Ez a Fájl menüpont Help-je." \
"2" "Szerkesztés" "Ez a Szerkesztés menüpont Help-je." \
"3" "Eszközök" "Ez az Eszközök menüpont Help-je." \
"4" "Beállítások" "Ez a Beállítások menüpont Help-je." \
"5" "Kilépés" "Vigyázat, kilépés ! "`

case $valasz in
1)

    until test
    do

    valasz=`Xdialog --stdout \
    --title "Shell Programozás 5/3 script" \
    --backtitle "Menü Próba" \
    --no-cancel --no-tags --item-help \
    --menubox "Válasszon a menüből." 0 0 5 \
    "1" "Megnyitás" "Ez a Megnyitás menüpont Help-je." \
    "2" "Mentés" "Ez a Mentés menüpont Help-je." \
    "3" "Vissza..." "Visszalépés egy menüvel fentebb."`

    case $valasz in
```

```
    1)
        fajl=`Xdialog --stdout \
        --title "Shell Programozás 5/3 script" \
        --backtitle "Ön egy már meglévő dokumentumot \
        akar kinyitni." --no-buttons \
        --help "Kár bármit is megnyitni\nez csak egy test script."\
        --fselect "" 0 0`

        Xdialog --title "Shell Programozás 5/3 script" \
        --msgbox "Most a $fajl fájl megnyitása történne, \
        ha ez egy éles program lenne." 0 0
    ;;
    2)
        fajl=`Xdialog --title "Shell Programozás 5/3 script"\
        --backtitle "Ön dokumentumot akar elmenteni\nAdja meg \
        hová és milyen néven." --help "Ez csak egy test \
        script\nValójában nem ment semmit." --fselect "" 0 0`

        Xdialog --title "Shell Programozás 5/3 script" \
        --msgbox "Most a fájl, $fajl néven történő mentése \
        következne, ha ez egy éles program lenne." 0 0
    ;;
    3)
        break
    ;;

    esac

    done

;;
2)
    until test
    do

    valasz=`Xdialog --stdout \
    --title "Shell Programozás 5/3 script" \
    --backtitle "Menü Próba" --no-cancel --no-tags \
    --item-help --menubox "Válasszon a menüből." 0 0 5 \
    "1" "Kivágás" "Ez a Kivágás menüpont Help-je." \
    "2" "Másolás" "Ez a Másolás menüpont Help-je." \
    "3" "Beillesztés" "Ez a Beillesztés menüpont Help-je." \
    "4" "Vissza..." "Visszalépés egy menüvel fentebb."`

    case $valasz in
    1)
        Xdialog --title "Shell Programozás 5/3 script" \
        --msgbox "Ön kivágott egy részt." 0 0
    ;;
```

```
    2)
        # Amennyiben a főmenü 2. pontjának,
        #2. almenüjét választotta.
        Xdialog --title "Shell Programozás 5/3 script" \
        --msgbox "Ön kimásolt egy részt." 0 0
    ;;
    3)
        # Amennyiben a főmenü 2. pontjának,
        #3. almenüjét választotta.
        Xdialog --title "Shell Programozás 5/3 script" \
        --msgbox "Ön egy részt illesztett be." 0 0
    ;;
    4)
        break
    ;;

    esac

    done
;;
3)
        Xdialog --title "Shell Programozás 5/3 script" \
        --msgbox "Ön az Eszközök Menüpontot választotta." 0 0
;;
4)
        Xdialog --title "Shell Programozás 5/3 script" \
        --msgbox "Ön az Beállítások Menüpontot választotta." 0 0
;;
5)
        Xdialog --title "Shell Programozás 5/3 script" \
        --yesno "Kilép ?" 0 0
        v=$?
        if test $v -eq 0
        then
            exit
        fi
;;

esac

done
```

Itt a tényleges műveleteket végző sorokat, a case-ken belül kellene elhelyezni. Ha ezek sok sorból állnak az egész kód nehezen átlátható válik. Erre, de nem csak erre használható a scripten belüli saját függvények lehetősége, melyet a 7. részben tárgyalunk.

6. Ciklusok

while/until; test; do; done

Következő lehetőségünk amivel a scriptunk struktúráját, hatékonnyá tehetjük, a ciklusok használata. Sokszor van, hogy ugyan azt a műveletet, sokszor kell végrehajtani, vagy több változón, illetve fájlon. Erre a while/until és a for ciklusokkal lehet megoldást találni.
Szintaxisa:

while test <feltétel>
do
  <utasítások>
done

A feltétel vizsgálatra kerül, ha igaz akkor a do és a done közötti utasítások végrehajtódnak, majd a shell visszatér a feltételvizsgálathoz, ha igaz ismét végrehajtja az utasítás sort, mindezt egészen addig, amíg a feltétel igaz. Ha hamissá válik, akkor a done után folytatódik a script.
Másik változata az until, addig folytatja a ciklust, míg a feltétel hamis és amikor igazzá válik, akkor lép ki a ciklusból.

until test <feltétel>
do
  <utasítások>
done

A két megoldás ugyanazt eredményezi.
while test <feltétel>
until test ! <feltétel>

Gyakran van, hogy egy adott szakaszt, megadott alkalommal akarjuk egymás után futtatni, úgy, hogy közben nyomon követhető legyen éppen hanyadik alkalommal fut le. Más nyelvek, erre a for cilust használják, de az a shell-ben, mint alább majd látjuk egészen másképpen működik. Helyette, egy while vagy until ciklust használhatunk, egy változóval, aminek az értékét minden cilusban 1-el növeljük, így az tölti be a ciklusszámláló szerepét, a ciklusfutás megszakításának feltételében pedig maghatátozhatjuk, hogy ha ez a számláló elér egy értéket, a ciklus akkor fejezze be a futását. Erre Én álltalában az $i változót használom, több egymásba ágyazott ciklus esetén pedig $i1 $i2 stb.

Példaképpen nézzünk egy egyszerű esettet amely a begépelt szöveget megfordítja és így írja azt ki.

```
-----------------------------------------------------------------
#!/bin/bash
# 6.fejezet. 1.script

clear
read szoveg   # Megfordítandó szöveg bekérése

hosz=`echo $szoveg | wc -c`   # A szöveg hosszának lekérdezése.

# Annyiszor ismétlődik a ciklus, ahány karakterből a szöveg áll.
i="0"   # ciklusszámláló nullázása.
until test $hosz -eq $i
do
    let i=$i+1   # ciklusszámláló 1-el növelése.

    # A ciklus előre haladtával sorba kiveszi
    #1-1 karaktert a szövegből.
    k=`echo $szoveg | cut -c $i`

    # A fordított szöveget összeilleszti
    # karakterenként a ciklusok során.
    fordszov="$k$fordszov"
done

echo $fordszov   # A fordított szöveg kiírása.
-----------------------------------------------------------------
```

for; do; done

A for cilus a shell-ben egészen másképpen működik, mint a basic-ben és egyéb programnyelvekben az szokásos. Általában a for ciklus megadott számszor hajtódik végre.
A while/until ciklus esetén a megszakítást egy megadott feltételhez kötődik.
A shell for ciklus, a ciklusnak átadott értékeken egyenként végig megy, majd kilép a ciklusból.
Szintaxisa:

for in <kifejezés>
do
<utasítások>
done

Figyeljük meg a működés beli különbségeket.

```
$ atadott="Ez meg az és még ez is."

$ for atvett in Ez meg az és még ez is.; do echo $atvett; done
$ for atvett in "Ez meg az és még ez is."; do echo $atvett; done
$ for atvett in $atadott; do echo $atvett; done
$ for atvett in "$atadott"; do echo $atvett; done
```

A fenti utasítás sor, scripten belül így nézne ki:

```
for atvett in "$atadott"
do echo $atadott
done
```

Vagy:

```
for atvett in "$atadott"
do
echo $atadott
done
```

Érdekes, hogy scriptben a "do" után a parancs lehet új sorban, míg ha egy sorban írom le az egészet, akkor a "do" és a következő parancs között, nem lehet újsor karakter, vagyis egysoros parancs esetén ";" jel. Ez igaz a többi parancsra is, amikben a "do" utasítás szerepel, valamint az if;then;fi esetén is. (if; then ...), (while; do ...), stb.

Miután a fenti négy variációt végig próbáltuk, a for ciklusra, a következőket figyelhettük meg.
Az 1. esetben, az "in" után adott értékek, a szóközönként elválasztva külön-külön értékként vannak értelmezve és egyenként végrehajtódik velük a ciklus. A cikluson belül a "for" és az "in" okozott megadott, ebben az esetben $atvett nevű változóban lehet az átadott értékre hivatkozni.
A 2. esetben az átadott érték, egyetlen értékként lett értelmezve az idézőjelek miatt. A 3. és 4. esetben láthatjuk, hogy a fenti két megállapítás, akkor is igaznak bizonyul, ha az értékeket változóban adjuk át a ciklusnak.

A for ciklus nagy előnye, hogy file neveket is át tud venni. Az alábbi példa kilistázza az aktuális könyvtár fájljait.

```
$ for fajl in *; do echo $fajl; done
```

A regurális kifejezések is használhatók. Az alábbi sor csak az mp3 fájlokat listázza.

```
$ for fajl in *.mp3; do echo $fajl; done
```

A for működésére egy gyakorlatias példán keresztül mutatok rá. A működése során, már megismert utasításokat alkalmazunk.
Az alábbi script, az indításának könyvtárától, rekurzívan megkeresi az mp3 fájlokat, majd az adatait egy adatbázisba menti.
Működése a következő: beolvassa az könyvtárból és az alkönyvtárakból is az mp3 fájlokat, az mp3info programmal kiveszi belőlük az idtag adatokat, majd ezeket egy szöveges adatbázisba menti, a fájl teljes elérési útjával együtt. A működéséhez viszont egykét dolgot rendbe kell tenni. Ugyan is az mp3info nem minden idtag adatot ad ki külön sorba. A Title és Track, az Album és Year, valamint a Comment és Genre adatokat egy sorba adja ki. Ezért ezek közé a scriptben egy újsor karaktert helyezünk, hogy minden idtag adat, külön sorba helyezkedjen el.

A script felépítése a következő:

```
until test
do
      for in
      do
            if
            then
            else
            fi
      done
done
```

Az until egy végtelen ciklus, amiből akkor lépünk ki egy exit utasítással, amikor az if else ága végrehajtódik. Ekkor a script is befejezi a futását. A for ciklusban sorba bekérjük az adott mélységben található mp3 fájlokat. A for, ha nem talál a reguláris kifejezésnek megfelelő fájlt, akkor a változóba magát a reguláris kifejezést helyezi el. Ezt használjuk arra, hogy megállapítsuk, talált-e az adott mélységben mp3 fájlt. Ha igen, akkor az if, then ága hajtódik végre, ha nem akkor az else ág. A then ágban az idtag adatokat változókba helyezi és hozzáadja az adatbázishoz őket. Az else ágban számolja, hányadik mélységben nem talált már mp3 fájlt. Ha 5 egymás utáni mélységben nem talált, akkor az exit-el befejezi a script a futását. a for cilus után, mikor az az adott mélység mp3 fájlait átvizsgálta, egy mélységgel lentebb lép, a $keres változó értékének megváltoztatásával.

Nézzük konkrétan a scriptet.

```
-------------------------------------------------
#!/bin/bash
# 6.fejezet. 2.script

# a változók megfelelő kezdeti értékének beállítása.
# a numerikus adatként használt változók kezdeti értékének
# a 0-át be kell állítani,
# csak így lehet velük az első pillanattól számolni.

keres='*.mp3'   # az első mélységű keresési kifejezés
melyseg="0"
sorszam="0"
nincs="0"

# az induló könyvtár kiválasztása.
dir=`Xdialog --title "Shell Programozás 6/2 script" \
--no-buttons --dselect "Válassza ki a könyvtárat,\nahonnan\
kezdve listába akarja gyüjteni az mp3 fájlokat." 0 0`
```

```
#  Létrehozzuk az üres adatbázist, a megfelelő mezőinformációkkal.

echo '0;Előadó;Album;Év;Számsorszám;Számcím;Jegyzet;\
Stílus;Elérési út' > zenetár.dat

# mivel a feltétel megadása nélküli test parancs sosem lesz "igaz"
ezért ez az until esetén végtelen ciklust okoz.

until test
do
    for mp3fajl in $keres
    do
        # Megvizsgáljuk, adott mélységben talált-e mp3 fájlt.
        # Ha a for, $mp3fajl változóban a keresett ($keres)
        # reguláris kifejezést adja vissza, akkor nem talált.
        # A test "!" opciójával megfordítjuk a jelentést,
        # így ez akkor válik igazzá,
        # ha talált mp3 fájlt, azaz ha a $keres
        # és a $mp3fájl értékek nem egyenlőek.

        if test ! "$keres" = "$mp3fajl"
        then
            # a $sorszam valtozóban, számoljuk
            #a talált mp3 fájlokat, az adatbázisban
            # ezzel sorszámozzuk őket, ezt beírjuk az index mezőbe

            let sorszam=$sorszam+1
            echo $mp3fajl   # kiírjuk a megtalált mp3 fájlt,
                            # hogy érzékelhessük a script futását.

            # Az alábbi részben történik az mp3info álltal
            # adott idtag-ok mindegyikének külön-külön sorba
            # helyezése. Erre azért van szükség, hogy a különféle
            # idtag-okat, pontossan tudjuk kinyerni.
            # Az azonos sorban lévő idtag-ok esetén a második tag
            # elé egy újsor karaktert szúrunk. Ne felejtsük el,
            # az alábbi 5 sor, tulajdonképpen egyetlen parancssor,
            # "\" jelekkel tördelve.

            idtag=`mp3info "$mp3fajl" |\
            sed s/'Track:'/'\nTrack:'/g |\
            sed s/'Year:'/'\nYear:'/g |\
            sed s/'Genre:'/'\nGenre:'/g |\
            tr ';' ',' | tr '\n' '\;'`
            # A végén az összes újsor karaktert egy ";"
            # mezőelválasztóra cseréljük így mint egy több mezőből
            # álló egyetlen adatsort tudjuk kezelni.
            # De elötte az idtag adatokban a ";" karaktereket
            # ","-re cseréljük, hogy ha véletlen ";" karaktert is
            # tartalmaznak az idtag-ek, az ne tévessze meg
            # a mezőelválasztást.
```

```
# Az idtag-okat tartalmazó adatsorból a különféle
# idtag-eket, külön változókba helyezzük. Az első cut
# az adott igtag-et veszi ki, de ez még tartalmazza
# az idtag nevét is. pl.: "Title: Ez a szám címe"
# A második cut-al szétválasszuk az idtag nevétől
# az idtag tartalmat.

szamcim=`echo $idtag | cut -d';' -f 2 | cut -d: -f2-`
szamsorszam=`echo $idtag | cut -d';' -f 3 | \
cut -d: -f2-`

eloado=`echo $idtag | cut -d';' -f 4 | cut -d: -f2-`
album=`echo $idtag | cut -d';' -f 5 | cut -d: -f2-`
ev=`echo $idtag | cut -d';' -f 6 | cut -d: -f2-`
jegyzet=`echo $idtag | cut -d';' -f 7 | cut -d: -f2-`
stilus=`echo $idtag | cut -d';' -f 8 | cut -d: -f2-`

# az "-f 2-" jelentése a "-" miatt,:
# "a 2.mezőtől kezdve, az összes utána jövő is."
# Azért kell, mert elvileg az idtag is tartalmazhat
# ":" karaktert. és "-f 2" esetén,
# az utána jövő részt levágná.

# Létre hozzuk a beírandó adatsort, vagy rekordot.
# A $dir/$mp3fajl a teljes elérési utat adja meg.

sor="$sorszam;$eloado;$album;$ev;$szamsorszam;\
$szamcim;$jegyzet;$stilus;$dir/$mp3fajl"

# Majd hozzáfűzzük a adatbázis végéhez.
echo "$sor" >> zenetár.dat

nincs="0"
# Ez nullázza az mp3 fájlokat nem tartalmazó mélység
# számolását, ha még is talált egyet.
```

```
            else

            let nincs=$nincs+1
            # mp3 fájlokat nem tartalmazó mélység számolása.

            # Ha már egymásután 5 mélységben nem talált.
            if test $nincs -eq 5
            then
                exit   # kilépés a scriptből
            fi

        fi

    done

    # Ha az adott mélységet a for-al átvizsgálta,
    # a keresést eggyel mélyebbre irányítjuk.
    # 0. mélység: *.mp3
    # 1. mélység: */*.mp3
    # 2. mélység: */*/*.mp3 stb.

    keres='*/'"$keres"

done
```

---

7. Saját függvények készítése

Ebben a részben, a scripten belüli saját függvényekkel fogunk foglalkozni. Hosszabb scriptek megfelelő struktúráltságához, illetve a gazdaságos működéshez, elengedhetetlen ez a technika. Szintaxisa a következő:

```
function sajatfugveny ()
{
      <utasítások>
      return
}
```

A függvényeket a script elejére helyezzük. A függvény futását azon belül bármikor megszakíthatjuk a return paranccsal.
A függvény pont úgy viselkedik, mintha egy külső program lenne. Azaz lehet neki értékeket átadni a szokásos bemenetére, illetve fogadni is lehet a szokásos kimenetéről. Ennek fényében a következő megoldások is alkalmazhatók:

```
echo $valtozo | sajatfugveny | less
echo $valtozo | sajatfugveny > fajl
valtozo2=`echo $valtozo | sajatfugveny`
```

Mivel ezeknél a példáknál a szokásos kimenetet a képernyőről át irányítottuk máshová, ezért a scripten belüli egyetlen echo sem jelenik meg a képernyőn, hanem továbbítódik a megjelölt kimenetre. Illetve mivel a szokásos bemenetet is átirányítottuk, ezért a függvényen belüli read parancs is a megadott bemenetről olvas.

Lehet argumentumokat is átadni neki, amikre függvényen belül a $1, $2 $3 stb módon lehet hivatkozni. Ekkor ha a read parancsot használjuk, akkor a felhasználótól várja az adatbevitelt. De a függvényen belüli echo-k még mindig nem a képernyőre, hanem az átirányított kimenetre, a lenti esetben a "valtozo" nevű változóba íródnak.

```
valtozo=`sajatfugveny $arg1 $arg2 $arg3`
```

Ha a függvényen belül, szeretnénk a képernyőre írni akkor a szokásos kimenetet ne irányítsuk át, azaz a függvényt csak a nevére hivatkozva hívjuk meg, ha pedig a felhasználótól is akarunk adatot bekérni, akkor a bemenetet se irányítsuk át, hanem a függvénynek argumentumok formájában adjunk át értékeket.

```
sajatfugveny $arg1 $arg2 $arg3
```

Ebben az esetben viszont értéket a függvénytől csak a visszatérési érték, azaz a $? változón keresztül kaphatunk tőle, de ezen keresztül az érték csak egy 0 és 255 közötti számérték lehet. Ennek az értékét a függvényen belül a return parancs után adhatjuk meg. A retutn hatására a függvény befejezi a futását és a $? visszatérési értékbe a return után megadott érték kerül.

Ennek a korlátot úgy kerülhetjük ki, hogy a scripten belül meghatározunk egy változót, amelyet arra használunk, hogy az éppen futó függvény ezen keresztül tudjon bármilyen értéket átadni. Természetesen nem csak egy, hanem több is felhasználható. Lássunk erre egy példát. Ez a függvény a felhasználótól kérdez valamit, majd a válaszát adja vissza. Amit a $fe (függvény eredmény) változóban kapunk vissza a függvénytől .

```
-----------------------------------------------
#!/bin/bash
# 7.fejezet. 1.script

function dialogus ()
{
local valasz=""
local visszateres""
clear
echo -n "$1 "
read valasz
fe=$valasz
visszateres="10"
return $visszateres
}

kerdes="Mi a neved ?"
dialogus "$kerdes"
vissza=$?
b=$fe
echo "Üdvözlöm, $b"
echo "A függvény a $vissza értékkel tért vissza."
-----------------------------------------------
```

Felmerülhet a kérdés, hogy a függvényen kívül miért nem eleve a $valasz változóval jutunk hozzá a válaszhoz. Azért, mert akkor a függvényünk eleve egy komoly hibalehetőséggel rendelkezne. Mégpedig, hogy ha a függvényen belül is és a script törzsében is azonos, globális változókat használnánk, ez veszélyes eljárás, lenne, ugyanis több függvény egymásból történő hívása esetén ezek az értékek összekeveredhetnek, illetve a függvény könnyen felülírhatna egy a script törzsében szereplő változót. Ezért érdemes a függvényen belüli változókat elkülöníteni a script többi részében szereplőtől. Ezt úgy tehetjük meg, hogy a függvény elején a local kulcsszóval deklaráljuk a változót. Meghatározás közben már értéket is kaphat.

local valasz=""
local a="Érték"

Ebben az esetben lehet a függvényen kívül is, vagy más függvényekben is egy "valasz" nevű változónk, ezek mind külön-külön értékkel rendelkezhetnek és így az értékek egy hosszabb, vagy bonyolultabb script esetén sem keverednek össze. Ezért a függvényen belül használatos változókat érdemes a függvény elején lokálisnak meghatározni. Ez alól az képez kivételt, ha tudatosan használunk globálisan, pl a függvény és a script törzse közötti értékmozgatás céljából. A fenti példában éppen ezt használjuk fel az $fe változó esetén, miközben viszont a $valasz változót lokálisnak határozzuk meg benne.

Láthatjuk a példában, hogy a return után adtunk meg egy visszatérési értéket is a függvényben. Most csak azért, hogy láthassuk, ezt a lehetőség is.

Most tegyük a fenti dialógus függvényünket kicsit látványosabbá. De ehhez ismerkedjünk meg egy újabb paranccsal, a tput -al. A tput, beállít egy terminált vagy lekérdezi a terminfo-t. De mi most azt a lehetőségét használjuk ki, hogy segítségével, pozicionálni lehet a kurzort a képernyőn illetve le lehet kérdezni az adott terminál hány sorból és hány oszlopból áll.

`tput cols`
Vissza adja, hány karakter széles a terminálunk.

`tput lines`
Vissza adja, hány karakter magas a terminálunk.

`tput cup <$x> <$y>`
A kurzor pozíciót, a megadott koordinára helyezi.

Készítünk egy függvényt, amely egy dialógus "ablakot" jelenít meg a terminálon. Ehhez több segéd függvényt is készítünk. Lesz egy keret() nevű amely egy keretet rajzol a képernyőre. Lesz egy box-torol() amely törli a megadott négyzetben a képernyőt. Lesz egy inputbox() nevű, amely a kérdést tehetünk fel a felhasználónak és választ kaphatunk tőlle, végül pedig egy msgbox () függvény, amellyel egy üzenetet tudunk megjeleníteni megadott másodpercig.
Természetesen ezeken a függvényeken rengeteget lehetne finomítani. Például a dialógus függvények csak a terminál közepén helyezkednek majd el, természetesen meg lehetne adni, hogy egyénileg lehessen őket pozicionálni. Ezenkívül csak rövid, egy soros üzeneteket kezel. Ezen is lehetne finomítani, hogy ha túl hosszú az üzenet, akkor feldarabolja és több sorba jelenítse meg azt, a dialogusbox méretét pedig ehhez igazítsa. De most a célunk csak az, hogy bemutassuk a függvények használatát.
Most röviden ismertetem a függvények működését.

keret()
Meghívása a következő módon történik:
keret <sor-pozició> <oszlop-pozició> <magasság> <szélesség>

Ahol a <sor-pozició> és az <oszlop-pozició>, a megjelenítendő keret balfelső sarkának koordinátáját határozza meg. A <magasság> a keret magasságát, azaz, hogy hány sor magas legyen, a <szélesség> pedig a szélességét, azaz, hogy hány karakter széles legyen, határozza meg.
A függvény először előállít két stringet, amik a a keret megrajzolásához fog használni. Egyik a keret felső és alsó széle lesz, a másik a köztes részek. Például:

```
 boxszel="################"
boxkozep="#              #"
"################"
"#              #"
"#              #"
"#              #"
"################"
```

Ezeket a szélességből alapján meghatározott hosszra készíti el. A kirajzolást a sor-pozició és az oszlop-pozició alapján kezdi el. Hogy hány darab köztes részt rajzoljon, azt pedig a magasság alapján határozza meg. A kirajzolást egyszerű echo parancsak végzi, de elötte egy tput cup x y -al pozicionálja a kurzort. Visszatérési értéke nincs.

box-torol()
Meghívása a következő módon történik:
box-torol <sor-pozició> <oszlop-pozició> <magasság> <szélesség>
Működése hasonló a keret() függvénnyel, annyi különbséggel, hogy itt egy megfelelő hosszúságú space-eket tartalmazó string kerül kinyomtatásra, megfelelően pozicionálva és így törli az adott négyzet területet a képernyőn. Visszatérési értéke nincs.

msgbox()
Meghívása a következő módon történik:
msgbox <szöveg> <másodperc>
tput lines és tput cols parancsokkal lekérdezi, az aktuális terminál méretét. A kérdés hossza alapján meghatározza a szükséges keret szélességét, majd az ablak koordinátáit, ez és a terminál méret alapján meghatározza, hogy az középre kerüljön. A kiszámolt értékekkel meghívja a keret() függvényt. Miután az kirajzolta a keretet, a megfelelő pozícióba kiírja a megadott szöveg-et, majd sleep paranccsal felfüggeszti a meghatározott másodpercig a script futását, ezáltal időt adva az üzenet elolvasására. Ezután pedig meghívja a box-torol() függvényt a szükséges argumentumokkal az msgbox törléséhez.

inputbox ()
Meghívása a következő módon történik:
inputbox <kerdes>
Hasonló az msgbox() függvényhez, annyi különbséggel, hogy a keretméret kiszámolásakor, hagy helyet a válasz beolvasásához is, azaz magasabbra formálja. Meghívja a keret() függvényt, kiírja a kérdést, majd egy read-al választ vár rá a felhasználótól. Amikor az, enter-el elküldi a válaszát, akkor a $fe script-globális változóba helyezi a választ, hogy azt a függvényen kívül is el lehessen érni, majd a box-torol() függvénnyel törli a dialógusbox-ot.

Nos lássuk a scriptet kikommentezve:

```
-------------------------------------------------
#!/bin/bash
# 7.fejezet. 2.script

#====================================
# ITT KEZDŐDNEK A SAJÁT FÜGGVÉNYEK

#-----------------------------------------------------------------
--
# ITT KEZDŐDIK INPUTBOX FÜGGVÉNY
function inputbox ()
{
# A függvényben használt változók,
#lokálisként történő meghatározása.
local boxsor=""
local boxoszlop=""
local boxmagas=""
local boxszeles=""
local kerdes=$1
local valasz=""
local kerdhosz=""

# A terminál méretének lekérdezése.
termmagas=`tput lines`
termszeles=`tput cols`
7
kerdhosz=`echo "$kerdes" | wc -c | tr -d ' '`

# A box szélességének és magasságának meghatározása.
let boxszeles=$kerdhosz+4
boxmagas="7"

# A keret balfelső sarka koordinátáinak kiszámolása,
# hogy a box középre kerüljön.
let boxsor=$termmagas/2-$boxmagas/2
let boxoszlop=$termszeles/2-$boxszeles/2

# A keret() függvény meghívása
keret $boxsor $boxoszlop $boxmagas $boxszeles

# A kérdés kiíratás kezdő pozicióinak kiszámolása.
let sor=$boxsor+2
let oszlop=$boxoszlop+3

# Kurzor megfelelő helyre állítása és a kérdés kiírása.
tput cup $sor $oszlop
echo "$kerdes"
```

```
# A válasz bekérés kezdő pozícióinak kiszámolása.
let sor=$sor+2
let oszlop=$oszlop+2

# Kurzor megfelelő helyre állítása és a válasz beolvasása.
tput cup $sor $oszlop
read valasz

# A válasz $fe script-globális változóba írása.
fe=$valasz

# Az inputbox törlése.
box-torol $boxsor $boxoszlop $boxmagas $boxszeles

}
# ITT VÉGZŐDIK AZ INPUTBOX FÜGGVÉNY
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# ITT KEZDŐDIK MSGBOX FÜGGVÉNY
function msgbox ()
{
# A függvényben használt változók,
# lokálisként történő meghatározása.
local boxsor=""
local boxoszlop=""
local boxmagas=""
local boxszeles=""
local szoveg=$1
local szoveghosz=""
local mp=$2

# A terminál méretének lekérdezése.
termmagas=`tput lines`
termszeles=`tput cols`

# A kérdés hosszának lekérdezése
# és az eredményből a space-ek törlése.
szoveghosz=`echo "$szoveg" | wc -c | tr -d ' '`
let boxszeles=$szoveghosz+4
boxmagas="5"

# A keret bal-felső sarka koordinátáinak kiszámolása,
# hogy a box középre kerüljön.
let boxsor=$termmagas/2-$boxmagas/2
let boxoszlop=$termszeles/2-$boxszeles/2

# A keret() függvény meghhívása
keret $boxsor $boxoszlop $boxmagas $boxszeles
```

```
# A szöveg kiíratás kezdő pozícióinak kiszámolása.
let sor=$boxsor+2
let oszlop=$boxoszlop+3

# Kurzor megfelelő helyre állítása és a szöveg kiírása.
tput cup $sor $oszlop
echo "$szoveg"

# A meghatározott ideig történő várakozás,
# hogy a felhasználó elolvashassa az üzenetet.
sleep $mp

# Az msgbox törlése.
box-torol $boxsor $boxoszlop $boxmagas $boxszeles

}
# ITT VÉGZŐDIK AZ MSGBOX FÜGGVÉNY
#-----------------------------------------------------------------

#-----------------------------------------------------------------
# ITT KEZDŐDIK A KERET FÜGGVÉNY
function keret ()
{
# A függvényben használt változók,
# lokálisként történő meghatározása.
local boxsor=$1
local boxoszlop=$2
local boxmagas=$3
local boxszeles=$4
local k="#"
local boxsor2=""
local boxoszlop2=""
local sor="0"
local boxszel=""
local boxkozep=""
local i=""
local spacehossz=""

# A jobbalsó sarok koordinátáinak kiszámítása.
let boxsor2=$boxsor+$boxmagas-1
let boxoszlop2=boxoszlop+$boxszeles-1

# A keret alsó és felső részét alkotó string előállítása.
# A ciklus boksz-szélégszer fut le, minden ciklusban egy darab,
# a keretet kirajzolásához meghatározott
# karakterrel növelve a stringet.
# Az $i változóval számolva a ciklusfutást.
```

```
let i="0"
until test $i -eq $boxszeles
do
    let i=$i+1
    boxszel="$boxszel$k"
done

# A keret középső részeit alkotó string előállítása.
# A string első karakterének a keretet
# kirajzolásához meghatározott karaktert teszi meg,
boxkozep=$k

# majd a ciklus boxszélég-2 alkalommal,
# egy-egy space-el növeli a stringet,
let i="0"
let spacehossz=$boxszeles-2
until test $i -eq $spacehossz
do
    let i=$i+1
    boxkozep="$boxkozep "
done

# végül a végére is helyez egy kirajzoló karaktert.
boxkozep="$boxkozep$k"

# A kurzor, a keret felső szélének kirajzolásának
# pozíciójába állítása, és a szél kirajzolása.
tput cup $boxsor $boxoszlop
echo $boxszel

# A box belseje, felső és alsó sorának meghatározása.
let sor=$boxsor
let also=$boxsor2-1

# A box közepét alkotó string kiírása a megfelelő sorokba.
# Kezdve a $sor sorba és folytatva egyenként az alatta
# következőkbe,amíg el nem éri a szükséges alsó sort.
until test $sor -eq $also
do
    let sor=$sor+1
    tput cup $sor $boxoszlop
    echo "$boxkozep"
done

# A kurzor, a keret alsó szélének kirajzolásának
# pozíciójába állítása, és a szél kirajzolása.
tput cup $boxsor2 $boxoszlop
echo $boxszel
}
# ITT VÉGZŐDIK A KERET FÜGGVÉNY
#----------------------------------------------------------------
```

```
#-----------------------------------------------------------------
# ITT KEZDŐDIK A BOX-TOROL FÜGGVÉNY
function box-torol ()
{
# A függvényben használt változók, lokálisként
# történő meghatározása.
local boxsor=$1
local boxoszlop=$2
local boxmagas=$3
local boxszeles=$4
local sor="0"
local kozep=""
local i="0"

# A jobbalsó sarok koordinátáinak kiszámítása.
let boxsor2=$boxsor+$boxmagas-1
let boxoszlop2=$boxoszlop+$boxszeles-1

# A megfelelő hosszúságú, space-et tartalmazó string előállítása.
until test $i -eq $boxszeles
do
    let i=$i+1
    kozep="$kozep "
done

# A kirajzolt box, bal-felsősarka pozíciójától kezdve,
# a space-et tartalmazó stringel, a box törlése, annyi soron át,
# amilyen magas a box volt.
let sor=$boxsor-1
until test $sor -eq $boxsor2
do
    let sor=$sor+1
    tput cup $sor $boxoszlop
    echo "$kozep"
done

}
# ITT VÉGZŐDIK A BOX-TOROL FÜGGVÉNY
#-----------------------------------------------------------------

# ITT VÉGZŐDNEK A SAJÁT FÜGGVÉNYEK
#====================================
```

```
#====================================
# ITT KEZDŐDIK A SCRIPT TÖRZSE
# A VÉGREHAJTÁS ITT KEZDŐDIK EL.

# Képernyő törlés
clear

# A kérdés meghatározása.
kerdes="     Mi az Ön neve ?       "

# Az inputbox meghívása a kérdéssel.
inputbox "$kerdes"

# Az inputbox() függvényben a válasszal feltöltött
# $fe script-globális változóból,a válasz másik változóba írása.
nev=$fe

# A s visszajelző szöveg meghatározása.
szoveg="    Üdvözlöm kedves $nev !    "

# A szöveg msgbox() függvénnyel történő kiíratása,
# a meghatározott másodpercig fent tartott msgboxban.
msgbox "$szoveg" 3

# A kurzor, a képernyő bal-felső sarkába állítása.
tput cup 0 0

# ITT VÉGZŐDIK A SCRIPT TÖRZSE
#====================================
```

A script első pillantásra hosszú. De ezentúl elég csak a már meglévő fogvényeket az új scriptünk elejére másolni és már is használhatjuk őket. Hiszen a fenti sriptben is amit le kell programozni, ha már rendelkezünk a függvény résszel, csak 7 sor !!!
Ennél már csak az lenne jobb, ha nem is kellene a scriptjeink elejére másolni, hanem lenne egy külső függvénykönyvtár, amiből elérhetjük azt. Ehhez előbb készítsük el a scriptet, amit meghíva, hozzáférhetünk a függvényeinkhez. Másoljuk a fenti scriptből, a függvényeket egy másik üres script fájlba. (Azaz #!/bin/bash -al kezdődő, futási joggal rendelkező fájlba.) Mondjuk legyen a neve "sfugv". Ezután fűzzük a script végére az alábbi részt, majd a scriptet másoljuk az /usr/bin/ könyvtárba, ami a $PATH része kell legyen. A külső fügvényscriptet tetszőleges mennyiségű sajátfüdvénnyel ővíthetjük, csak ne felejtsük el a alábbi szakaszban is a rá vonatkozó új case szakaszt létrehozni.

```
-------------------------------------------------
case $1 in
keret) keret $2 $3 $4 $5 ;;
box-torol) box-torol $2 $3 $4 $5 ;;
inputbox) inputbox "$2" ;;
msgbox) msgbox "$2" $3 ;;
*) msgbox "Ismeretlen függvényhívás" 5
esac
echo $fe 1>&2
```

Az eredeti scriptünk helyett, pedig hozzunk létre egy másikatt, aminek az alábbi legyen a tartalma:

```
-------------------------------------------------
#!/bin/bash
# 7.fejezet. 3.script

clear
kerdes="     Mi az Ön neve ?       "
sfugv inputbox "$kerdes" 2> /tmp/$0.$$
nev=`cat /tmp/$0.$$`
szoveg="    Üdvözlöm kedves $nev !   "
sfugv msgbox "$szoveg" 3
tput cup 0 0
-------------------------------------------------
```

A következő képen működik a dolog. Az sfugv scriptünket meghívjuk, első argumentumként megadva neki, hogy melyik függvényt szeretnénk használni, utána pedig a többi argumentumban, ami adatokat a függvénynek akarunk átadni. Az sfugv script, az első argumentumban kapott érték alapján, case-vel megnézi melyik függvényt kell futtatnia, annak átadja a többi argumentumot amit kapott és lefuttatja a függvényt, majd ami értéket, (ha van ilyen,) vissza adott a függvény, a sfugv scripten belüli globális $fe változóba, azt a stderr azaz a szokásos hiba kimenetre küldi. Az eredeti függvényben, amiből meg hívtuk a sfugv scriptet, a függvény hibakimenetét át irányítjuk egy fájlba, majd a fájlból cat segítségével helyezzük változóba. Természetesen azoknál a függvényeknél nincs erre szükség, amik nem adnak vissza értéket. Amik pedig vissza adnak, azoknál viszont azért van rá szükség, mert ha a sfugv szokásos kimenetét átirányítjuk, pl egy változóba, akkor a sfugv script összes függvényének összes echo-jat oda írja, nem pedig a képernyőre, vagyis akkor a függvények nem tudnának kommunikálni a felhasználóval. Így viszont amit a felhasználónak üzen, azt a szokásos kimenetre írja, viszont a vissza adott értéket a hibakimenetre. Így nem keveredik a kettő össze. Innen viszont egy átmeneti fájlon keresztül tudjuk változóba tenni.

Még szót ejtenék a $0.$$ nevű ideiglenes fájlról. Ez mindig egy egyéni nevű fájlt hoz létre, vagyis, ha egyszerre több process, használja a scriptet, akkor sem keverednek össze az eredmények, mint akkor történne, ha fix nevű fájlt használnánk. Ez a fájl elnevezés esetén a $0 változó, a futó script nevét, a .$$ rész pedig a futó processz számát adja a fájl nevébe. Azaz egy myscript nevű script futása esetén, amely pl. a 6234-es pidszám alatt fut, myscript.6234 lesz az ideiglenes fájl neve. De ezzel nem kell törődnünk, hiszen mikor ki olvassuk a fáfl tartalmát, akkor is elég a $0.$$ néven hivatkozni rá.

A saját inputbox() függvény írásán keresztül, jobban megérthettük az Xdialog működését is. Alap helyzetben az is a hibakimenetre ír, de nála adva van egy --stdout kapcsoló, aminek a hatására a normál kimenetre adja vissza az Xdialog boxok álltal visszaadott értékeket. Bár ha belenézünk az Xdialog-hoz kapott samples, vagyis példa scriptekbe, (/usr/share/doc/Xdialog/samples vagy /usr/local/share/doc/Xdialog/samples,) akkor láthatjuk, hogy a legtöbb esetben ott is egy ideiglenes fájlon keresztül oldják meg az érték visszaadása.

Xdialog példaként itt arra láthatunk egy lehetőséget, hogyan lehet több dialógusablakot is nyitvatartani egyszerre. Tulajdonképpen külön függvényekbe helyezzük a dialógusablakok indítását és a függvényt a háttérben futtatjuk. Ezt ugyanúgy kell tenni, mint bármilyen program esetében. "fugvenynev &". Természetesen pusztán a több ablak futtatása úgy is lehetséges, ha fügvény nélkül, a dialogbox-ot indító parancs után teszünk "&" jelet, de azért a függvényben való elhelyezés a jó megoldás, mert csak akkor van lehetőség, a dialogbox visszatérési értékének feldolgozására is.

```
---------------------------------------------------------------------
#!/bin/bash
# 7.fejezet. 4.script

function ablak1 ()
{
Xdialog --stdout --title "Shell Programozás 7/4 script" \
--backtitle "Első ablak" \
--3rangesbox "Kérem állítsa be az értékeket" 0 0 \
"Biztonság" "1" "10" "5" \
"Gyorsaság" "1" "1000" "300" \
"Stabilitás" "1" "100" "80"
}

function ablak2 ()
{
Xdialog --stdout --title "Shell Programozás 7/4 script" \
--backtitle "Második ablak" --calendar "IDŐGÉP :-)" 0 0 \
"24" "07" "2381"
}
```

```
function ablak3 ()
{
Xdialog --stdout --title "Shell Programozás 7/4 script" \
--backtitle "Harmadik ablak" --item-help \
--buildlist "Válassza ki a megfelelő neveket." 0 0 "10" \
"1" "Kovács Béla" "on" "Kecskemét" \
"2" "Kormos Katalin" "off" "Baja" \
"3" "Sípos Antal" "on" "Budapest" \
"4" "Baranyi Andrea" "on" "Zalaegerszeg" \
"5" "Németh Balázs" "off" "Pécs"
}

ablak1 &
ablak2 &
ablak3 &
Xdialog --stdout --backtitle "" \
--title "Shell Programozás 7/4 script" \
--msgbox "Befejezze a script a futását ?" 0 0

killall Xdialog
```

---

8. Tömbváltozók és többdimenziós tömbök modellezése

A bash shell ismeri a tömb típusú változókat. Nagy hátrány, hogy csak egydimenziósakat ismer. Azaz a többdimenziós tömböket nem ismeri. Nézzük a tömb változó kezelését.

Meghatározás, vagy deklarálás:
$ declare -a valt[elemszam]

Dinamikus tömbökről van szó, azaz a meghatározott elemszám nem köt, hanem túl lehet rajta lépni. Deklarálni sem szükséges, hanem mint a normál változónál, amikor a tömbnek vagy annak egy elemének értéket adunk, akkor az létrejön. A tömb első eleme a 0 sorszámú.

Értékadás:
tomb[elemszám]="érték"

```
$ tomb[3]="Szabó Julianna"
```

Érték kinyerése:
echo ${tomb[elemsorszám]}

```
$ echo ${tomb[3]}
Szabó Julianna
$ b=`echo ${tomb[3]}`
```

A tömb típusú változóknak az is az előnye. hogy ciklusokban könnyen kezelhetőek. A ciklus számlálóval hivatkozunk a tömb egyes elemeire.

```
i="0"
until test $i -eq 4
do
   let i=$i+1
   echo ${tomb[$i]}
done
```

Egy tömböt a nullázni, vagyis minden elemét üressé tenni a következőképpen lehet:

```
$ tomb=()
```

A kétdimenziós tömböket, egy excel táblához hasonlíthatjuk, aminek az egyik dimenzióját a sorok, a másik dimenzióját az oszlopok adják. Egy elemre, pedig hasonlóan hivatkozhatunk, mint a excel tábla egy cellájára, azaz annál megadjuk a sort és az oszlopot ami kereszteződésében a cella van, itt pedig megadjuk, az első dimenzió elemsorszámot és a második dimenzió elemsorszámot.

Két dimenziós tömböt úgy is, modellezhetjünk, ha az egydimenziós tömb elemeibe egy-egy adatsort teszünk, aminek a mezői képezik a második dimenziót. Ezeket, a már ismert módon, mint mezőkre bontott adatsorokat kezeljük. Az első dimenzió irányát, a tömb elemek alkotják amik tulajdonképpen adatsorok, a másodikét az egyes tömbelemekben tárolt további elemek, vagyis az adatsorok mezői. A nulladik elemben, tárolhatjuk a mező neveket. Azaz a kétdimenziós tömbünk, logikailag pontosan úgy fog kinézni, mint a adatok.dat fájlunk. A 0. sorszámú tömbelem első mezőjét, ami a adatok.dat fájl "Sorszám" mezőoszlop 0. sorának felel meg, felhasználhatjuk a fájl nevének tárolására, amiből az adatokat betöltöttük.
Ez a technika a két dimenziós tömböknél még alkalmazható, de a három, vagy több dimenziósoknál már nem. Bár ezeket ritkán használjuk.

Az alábbi script, az adatok.dat adatbázis fájl tartalmát listázza ki, adatsoronként, azon belül pedig mezőkre bontva azt.

```
-------------------------------------------------------
#!/bin/bash
# 8.fejezet. 1.script

i="-1"
let sorokszama=`cat adatok.dat | wc -l | tr -d ' '`-1

until test $i -eq $sorokszama
do
    let i=$i+1
    a[$i]=`cat adatok.dat | sed -n "$((i+1)) p"`
    mezodb=`echo "${a[$i]}" | tr ';' '\n' | wc -l | tr -d ' '`

    echo "########################################"
    echo "####### ${i}-dik adatsor feldolgozása #######"
    echo "########################################"

    i2="0"
    until test $i2 -eq $mezodb
    do
        let i2=$i2+1
        mezonev=`echo "${a[0]}" | cut -d';' -f $i2`
        mezo=`echo "${a[$i]}" | cut -d';' -f $i2`
        echo "$mezonev : $mezo"
    done

done
-------------------------------------------------------
```

Mivel ezzel a módszerrel, három és több dimenziójú tömböket nincs értelme modellezni, ezért nézzük meg, hogyan modellezhetünk másképpen többdimenziós tömböket. Előbb az egydimenziós, majd a többdimenziós tömb modellezését, de már elszakadva az adatbázis kezelés példájától.

Tudjuk, hogy a shell változói, alaphelyzetben string típusú változók. Ha meghatározom, hogy a tömb elemeit milyen karakterrel határolom el egymástól, akkor egy szöveges változóban is tárolható egy teljes tömb. Ebből pedig a cut paranccsal kinyerhető a szükséges tömb elem.
Nézzünk egy példát, egy neveket tartalmazó, egydimenziós tömböt. Legyen a tömbelem elválasztó karakter a megszokott ";".

```
$ tomb="első elem;második elem;harmadik elem;\
negyedik elem;ötödik elem;hatodik elem"

$ elem1=`echo $tomb | cut -d';' -f1`
$ elem2=`echo $tomb | cut -d';' -f2`
$ elem3=`echo $tomb | cut -d';' -f3`
$ elem4=`echo $tomb | cut -d';' -f4`
$ elem5=`echo $tomb | cut -d';' -f5`
$ elem6=`echo $tomb | cut -d';' -f6`
```

Tehát a tömb adott sorszámú elemére így tudunk hivatkozni.

```
$ elemsorszam="3"
$ elem=`echo $tomb | cut -d';' -f $elemsorszam`
$ echo $elem
harmadik elem
```

No de ha egy ciklusban szeretnénk felhasználni a tömbelemeket, akkor tudnunk kell, hány elemből is áll a tömb. Ehhez nézzük meg, hány alkalommal fordul elő benne, a tömbelem elválasztó ";" karakter. Erre a wc programot használjuk -l kapcsolóval, ami vissza adja hány sorból is áll a bemenetére küldött adatsor. De mivel a "wc -l" a sorokat számolja meg, ezért eljötte a ";" jeleket, sorvégjelekre "\n" cseréljük, majd utána a wc által adott számról levágjuk a felesleges szóközöket.

```
$ tombelemdb=`echo $tomb | tr ';' '\n' | wc -l | tr -d ' '`
$ echo $tombelemdb
6
```

A tömb elemeinek feldolgozása egy ciklusban:

```
tombelemdb=`echo $tomb | tr ';' '\n' | wc -l | tr -d ' '`
i="0"
until test $i -eq $tombelemdb
do
    let i=$i+1
    echo $tomb | cut -d';' -f $i
done
```

Mindez nem túl ismeretlen, hiszen az előző részekben már használtuk ezt a technikát, más részt, ez a bash által is ismert egydimenziós tömbbel egyszerűbben megoldható. De tovább fejlesztve ezt, akár a lényegesebb, tényleges programozási nyelvekben ismert, tömbkezelő függvények is le modellezhetők.

Készítsünk egy függvényt, ami kiírja a tömb általunk meghatározott sorszámú elemét. A függvénynek az első argumentumban a tömböt kell átadni, a másodikban a kívánt elem sorszámát. A függvény az elemet $fe script-globális változóba adja vissza. Senkit ne zavarjon meg a sorok elején lévő "local" deklaráció. Csupán nem a függvény elején határozzuk meg a belső változók lokális voltát, hanem akkor, amikor értéket kapnak.

```
function tomb1d_kiir ()
{
local tombtorzs=$1
local elemsorszam=$2
local elem=`echo $tombtorzs | cut -d';' -f $elemsorszam`
fe=$elem
}
```

Használata:

```
index=3
tomb1d_kiir $tomb $index
elem=$fe
```

Aki a http://glindorf.fw.hu -ról töltötte le a saját fügvények, "sfugv", vagy „sfugv+” scriptet és azt másolta be a /bin könyvtárába, annak így is elérhető:

```
sfugv "tomb1d_kiir" "$tomb" $index 2> /tmp/$0.$$
elem=`cat /tmp/$0.$$`
```

Most készítsünk egy olyan függvényt, ami az adott tömbelemet értékkel tölt fel. A függvénynek a tömb és a feltöltendő elem sorszáma mellett, a feltöltendő tartalmat is át kell adni. Ehhez a tömbből külön kell választani, az értékkel feltöltendő elem előtti szakaszt és az az utáni szakaszt, majd összeilleszteni őket, de közéjük szúrva az értékkel feltöltött elemet. Az új tömböt a $fe script-globális változóba adja vissza.

```
function tomb1d_feltolt ()
{
local tombtorzs=$1
local elemsorszam=$2
local tartalom="$3"

# A tömb elemszámának megnézése.
local maxelem=`echo $tombtorzs | tr ';' '\n' | wc -l | tr -d ' '`
# Ha az első elem kell, akkor nem kell
# első levágott rész, az az, az legyen üres.
local ig=$((elemsorszam-1))
if test $ig -ge 1
then
   local tombeleje=`echo $tombtorzs | cut -d';' -f 1-$ig`
else
   local tombeleje=""
fi
```

```
# Ha az utolsó elem kell, akkor nem kell
# utolsó levágott rész, az az, az legyen üres.
local tol=$((elemsorszam+1))
if test $tol -le $maxelem
then
   local tombvege=`echo $tombtorzs | cut -d';' -f $tol-`
else
   local tombvege=""
fi

# Ha közbülső elem, értékét kellett változtatni
if test $elemsorszam -ne 1 -a $elemsorszam -ne $maxelem
then
   local ujtomb="$tombeleje;$tartalom;$tombvege"
fi

# Ha az első elem, értékét kellett változtatni
if test $elemsorszam -eq 1
then
   local ujtomb="$tartalom;$tombvege"
fi

# Ha az utolsó elem, értékét kellett változtatni
if test $elemsorszam -eq $maxelem
then
   local ujtomb="$tombeleje;$tartalom"
fi

fe=$ujtomb
}
```

Használata:

```
index=3
ujtartalom="új elem-tartalom"
tomb1d_feltolt $tomb $index "$ujtartalom"
tomb=$fe

sfugv "tomb1d_feltolt" "$tomb" $index "$ujtartalom" 2> /tmp/$0.$$
tomb=`cat /tmp/$0.$$`
```

Tekintsük meg az eredményt:

```
echo $tomb | tr ';' '\n'
```

Egy kis ügyeskedéssel, a tömbelemek sorba rendezése, tömbelem beszúrása, kivágása és a többi általános tömbkezelő függvény is elkészíthető.

Többdimenziós tömbök modellezése.

A sajnos bash shell nem képes több dimenziójú tömb változók kezelésére. Holott ez egy igen hatékony eszköz, a programozás során. Főként, ha adatbázis szerű alkalmazásról van szó, hiszen ott sok esetben az adattáblát, egy 2 dimenziós tömb változóba olvassák be, ahol az egyik dimenzió elemei az adatsorok a másiké pedig az adatmezők. Most nézzük, hogyan modellezhetünk, két, vagy többdimenziós tömböt, a bash egydimenziós tömbjeinek felhasználása nélkül.

Ahány dimenziós a tömb, annyi féle elválasztó karakterre van szükség, amiket természetesen a tömbelemek által tartalmazott adatokban nem szerepelhetnek. Például egy két dimenziós tömb esetén legyen a két dimenziót, vagy úgy is mondhatnám, az adatsorokat elválasztó jel a "#".

Ez alapján hozzunk létre egy kétdimenziós tömböt. Első dimenziója két indexelemmel a második három indexelemmel rendelkezzen. Az érték amiket az elemek tartalmaznak, legyenek az adott elem indexszámai betűvel kiírva.

```
tomb2d="egy-egy;egy-kettö;egy-három#kettő-egy;kettő-kettő;\
kettő-három#három-egy;három-kettő;három-három"
```

Ha a feltöltéskor használjuk a "\" parancssortördelés lehetőségét, mindjárt jobban átlátható mit is csinálunk.

```
tomb2d="\
egy-egy;egy-kettö;egy-három#\
kettő-egy;kettő-kettő;kettő-három#\
három-egy;három-kettő;három-három"
```

Készítsük el a fenti két függvény kétdimenziós tömböt kezelő változatait.

```
function tomb2d_kiir ()
{
local tombtorzs=$1
local dim1elemsorszam=$2
local dim2elemsorszam=$3
local elem=`echo $tombtorzs | cut -d'#' -f $dim1elemsorszam |\
cut -d';' -f $dim2elemsorszam`
fe=$elem
}
```

Használata:

```
index1=2
index2=3
tomb2d_kiir $tomb2d $index1 $index2
elem=$fe

sfugv "tomb2d_kiir" "$tomb2d" $index $index2 2> /tmp/$0.$$
elem=`cat /tmp/$0.$$`
```

Most pedig a feltöltést végző:

```
function tomb2d_feltolt ()
{
local tombtorzs=$1
local dim1elemsorszam=$2
local dim2elemsorszam=$3
local tartalom="$4"

#===== AZ ELSŐ DIMENZIÓ FELDARABOLÁSA =====
# Az első dimenzió (adatsorok) elemszámának megnézése.
local maxdim1=`echo $tombtorzs | tr '#' '\n' | wc -l | tr -d ' '`

# Ha az első elem (adatsor) kell, akkor nem kell
# első levágott rész, az az, az legyen üres.
local ig=$((dim1elemsorszam-1))
if test $ig -eq 0
then
   local dim1tombeleje=""
else
   local dim1tombeleje=`echo $tombtorzs | cut -d'#' -f 1-$ig`
fi

# Az elem, (adatsor).amely a változtatandó elemet
# (adatmezőt) tartalmazza
local adatsor=`echo $tombtorzs | cut -d'#' -f $dim1elemsorszam`

# Ha az utolsó elem (adatsor) kell, akkor nem kell
# utolsó levágott rész, az az, az legyen üres.
local tol=$((dim1elemsorszam+1))
if test $tol -le $maxdim1
then
   local dim1tombvege=`echo $tombtorzs | cut -d'#' -f $tol-`
else
   local dim1tombvege=""
fi

#===== AZ MÁSODIK DIMENZIÓ FELDARABOLÁSA =====
# A második dimenzió (adatmezők) elemszámának megnézése.
local maxdim2=`echo $adatsor | tr ';' '\n' | wc -l | tr -d ' '`

# Ha az első elem (adatmező) kell, akkor nem kell
# első levágott rész, az az, az legyen üres.
local ig=$((dim2elemsorszam-1))
if test $ig -ge 1
then
   local adatsoreleje=`echo $adatsor | cut -d';' -f 1-$ig`
else
   local adatsoreleje=""
fi
```

```
# Ha az utolsó elem (adatmező) kell, akkor nem kell
# utolsó levágott rész, az az, az legyen üres.
local tol=$((dim2elemsorszam+1))
if test $tol -le $maxdim2
then
   local adatsorvege=`echo $adatsor | cut -d';' -f $tol-`
else
   local adatsorvege=""
fi

#===== AZ ELSŐ DIMENZIÓ ADATSORÁNAK ÖSSZEILLESZTÉSE =====
# Ha közbülső elem, (adatmező) értékét kellett változtatni
if test $dim1elemsorszam -ne 1 -a $dim2elemsorszam -ne $maxdim2
then
   local ujadatsor="$adatsoreleje;$tartalom;$adatsorvege"
fi

# Ha az első elem, (adatmező) értékét kellett változtatni
if test $dim2elemsorszam -eq 1
then
   local ujadatsor="$tartalom;$adatsorvege"
fi

# Ha az utolsó elem, (adatmező) értékét kellett változtatni
if test $dim2elemsorszam -eq $maxdim2
then
   local ujadatsor="$adatsoreleje;$tartalom"
fi

#===== A TELJES KÉTDIMENZIÓS TÖMB ÖSSZEILLESZTÉSE =====
# Ha közbülső elem, (adatsor) egyik elemét,
# (adatmezőjét) kellett változtatni
if test $dim1elemsorszam -ne 1 -a $dim1elemsorszam -ne $maxdim1
then
   local ujtomb2d="$dim1tombeleje#$ujadatsor#$dim1tombvege"
fi

# Ha az első elem, (adatsor) egyik elemét,
# (adatmezőjét) kellett változtatni
if test $dim1elemsorszam -eq 1
then
   local ujtomb2d="$ujadatsor#$dim1tombvege"
fi

# Ha az utolsó elem, (adatsor) egyik elemét,
# (adatmezőjét) kellett változtatni
if test $dim1elemsorszam -eq $maxdim1
then
   local ujtomb2d="$dim1tombeleje#$ujadatsor"
fi
```

```
#===== AZ ÚJ TÖMB ÁTADÁSA A $fe GLOBÁLIS VÁLTOZÓBAN =====
fe=$ujtomb2d
}
-------------------------------------------------------------
```

Használata:

```
index1=2
index2=3
ujtartalom="új elem-tartalom"
tomb2d_kiir $tomb2d $index1 $index2 $ujtartalom
tomb2d=$fe

sfugv "tomb2d_feltolt" "$tomb2d" $index $index2 \
"$ujtartalom" 2> /tmp/$0.$$
tomb2d=`cat /tmp/$0.$$`
```

Tekintsük meg az eredményt:

```
echo $tomb2d | tr '#' '\n'
```

Természetesen ugyanígy modellez hetünk és használhatunk, három dimenziós tömböket is. Bár felmerülhet a kérdés, azoknak mikor is láthatjuk hasznát ? Például, lehetséges vele több adattáblás adatbázisok egyetlen változóba való olvasása és kezelése.
Mint a kétdimenziós tömböt egy excel táblához, úgy a három dimenziósat, egy több munkalapos excel táblázathoz hasonlíthatjuk, ahol a első dimenziót, a munkalapok alkotják, a másodikat a munkalapok sorai, a harmadikat pedig a munkalapok oszlopai alkotják.
Az adattáblák külön fájlokban is elhelyezkedhetnek és egy index fájl felsorolhatja öket.
Az adattáblák első mezőoszlopa az adatsorok sorszámozását, az első adatsora pedig az adatmezők neveit, illetve az első adatsor első adatmezője pedig az adattábla nevét is tartalmazhatja.

A többtáblás adatbázis, nem csak több fájlal, hanem egy táblahatároló karakter használatával, egyetlen fájlba is tárolható. Hozzunk létre egy Kecskeméti barátokat, Budapesti rokonokat és Szegedi kollégákat tároló több táblás adatbázist eggyetlen háromdimenziós tömbben. A "\" karaktert használtam a parancssor tördelésére, az átláthatóság kedvéért. Ezt nyugodtan így ahogy van bemásolhatjuk egy terminálba. A harmadik dimenzióelemeit, azaz a másodikdimenzió rétegeit, a „&” jellel szeparáljuk el.

```
$ tomb3d="\
barátok-tábla;barát-név;barát-cím#\
1;Barát Béla;Kecskemét#\
2;Barát Gábor;Kecskemét\
&\
rokonok-tábla;rokon-név;rokon-cím#\
1;Rokon Andrea;Budapest#\
2;Rokon Gergő;Budapest\
&\
kollégák-tabla;kolléga-név;kolléga-cím#\
1;Kolléga Ferenc;Szeged#\
2;Kolléga Katalin;Szeged\
"
```

Ezután írassuk ki, az adattáblákat egy üres sorral elválasztva, az adattábla adatsorait pedig külön-külön sorokba helyezve:

```
$ echo $tomb3d | sed s/'&'/'\n\n'/g | tr '#' '\n'
barátok-tábla;barát-név;barát-cím
1;Barát Béla;Kecskemét
2;Barát Gábor;Kecskemét

rokonok-tábla;rokon-név;rokon-cím
1;Rokon Andrea;Budapest
2;Rokon Gergő;Budapest

kollégák-tabla;kolléga-név;kolléga-cím
1;Kolléga Ferenc;Szeged
2;Kolléga Katalin;Szeged
```

Erre a területre már nem szándékozom jobban kitérni, de azért lássunk egy háromdimenziós tömb meghatározott elemét kiíró függvényt:

```
function tomb3d_kiir ()
{
local d3=$1
local d2=`echo $d3 | cut -d'ß' -f $2`
local d1=`echo $d2 | cut -d'#' -f $3`
local d0=`echo $d1 | cut -d';' -f $4`
fe=$d0
}
```

Használata:

```
tomb3d_kiir $tomb $index1 $index2 $index3
elem=$fe

sfugv "tomb3d_kiir" "$tomb2d" $index $index2 $index3 2> /tmp/$0.$$
elem=`cat /tmp/$0.$$`
```

Jelenlegi ismereteinket alapul véve, akár egy dbase szerű adatbáziskezelőt-scriptet is írhatnánk.Mivel a mező és dimenzió határoló karakterek nem szerepelhetnek a tartalomban, ezért érdemes olyanokat választani, amik nem igen használatosak és az adatbázis beolvasásakor és adatbevitelkor le kell ellenőrizni nincsenek-e benne olyanok.

Xdialog példa:

Most nézzük meg, hogy milyen hatékonyan lehet a tömböket és az Xdilaog lehetőségeit használni. Ezzel a scriptel egy az adatok.dat adatbázisunkhoz, vagy azzal azonos felépítésű adatbázishoz lehet további adatsorokat hozzáadni. A script nem végzi el a bevitt adatok formájának helyességét de természetesen ez megoldható. Megoldás lehetne, ha az adatbázisban a mezőnevek mellett azt is tárolnánk, milyen megszabott értékek kerülhetnek bele. Majd a script ezeket kiolvasná és az ilyen mezők esetében az adatbevitelt egy választó listából tenné lehetővé, az Xdialog --combobox -al.

```
-------------------------------------------------------
#!/bin/bash
# 8.fejezet. 2.script

# Szerkesztendő adatfájl kiválasztása
datfile=`Xdialog --title "Shell Programozás 8/2 script" \
--backtitle "Válaszd ki az adatbázist, \
ammihez további adatokat szeretnél hozzáfüzni ! " \
--stdout --no-buttons --fselect "" 0 0`
gomb=$?

# Ha nem igen gomb lett nyomva, akkor kilép a scriptből,
# akkor is, ha az ablak be lett zárva.
if test ! $gomb -eq 0
then
    exit
fi

# Az új adatsor sorszámának lekérdezése.
ujadatsorszama=`cat $datfile | wc -l | tr -d ' '`

# A nulladik adatsor, azaz a fájl első sorának beolvasása
# Esetünkben ez tartalmazza az adatmezők neveit
mezosor=`cat $datfile | sed -n '1 p'`

# Mezők megszámolása
mezodb=`echo $mezosor | tr ';' '\n' | wc -l | tr -d ' '`
```

```
# Az egész egy nagy ciklusban van, hogy több adatsort is
# fel lehessen vinni.A ciklusból akkor lép ki, ha a felhasználó
# nem akar több adatsort hozzáadni.
ki="0"
until test ! $ki -eq 0
do

# A cikluson belüli tömbök tartalmának törlés
mezo=()
prev=()

# Ez a belső ciklus sorbaveszi a mezőket és
# beolvassa a mezőneveket egy tombe
    i="0"
    until test $i -eq $mezodb
    do
        i=$((i+1))

        # Aktuális mező nevének lekérdezése
        mezonev[$i]=`echo $mezosor | cut -d';' -f $i`
    done

# Ezt a két speciális sort a két mezőbe bevihető értékekről
# tájékoztatja a felhasználót. Ezzel elveszti a script az
# univerzitását és csak az ilyen mezőkkel rendelkező
# adatbázisokhoz lesz jó.
mezonev[7]="${mezonev[7]} ( férfi / nő )"
mezonev[8]="${mezonev[8]} ( független / házas / elvált / özvegy )"

# Ez a ciklus újra sorba veszi a mezőket és bekéri az újmező
# tartalmát a felhasználótól. Közben a $bevittadat változóból
# tájékoztatja az addig bevitt adatokról. Az $ujadatsor változóba
# gyújtjük folyamatosan a már felvitt mezőadatokat.
i="1"
bevittadat=""
ujadatsor="$ujadatsorszama"
until test $i -eq $mezodb
do

# Ha az első mezőnél lett "Vissza" gomb nyomva az $i 0 lenne.
# Ezt akadájozza meg.
if test $i -eq 0
then
    i=1
fi
```

```
# A ciklus számlálása
i=$((i+1))
mezo[$i]=`Xdialog --stdout --title "Shell Programozás 8/2 script"\
 --backtitle\ "Adatfelvitel\n----------------------\nEddig\
 felvitt adatok:\n$bevittadat\n" \
--wizard --inputbox "${mezonev[$i]}" 0 0`
gomb=$?

# A megnyomott gombok szerinti végrehajtás
case $gomb in

# Igen gomb
0)
# Az eddig bevitt mezőadatokhoz az mostani ciklus adatának
# hozzáírása. A $bevittadatok változónál a mező nevét is
# hozzáírjuk és egy "\n" jellel az Xdialog számára sortörést is
# teszünk bele, hogy a mezőket külön-külön sorokba írja ki.
bevittadat="$bevittadat\n${mezonev[$i]} : ${mezo[$i]}"

# Az adatsor esetében a ";" mezőelválasztó karaktert tesz közéjük.
ujadatsor="$ujadatsor;${mezo[$i]}"

# Elmenti az aktuális ciklus adatait
bevittadatprev[$i]=$bevittadat
ujadatsorprev[$i]=$ujadatsor
;;

# Mégse gomb
1)

# Az adatsor feltöltő ciklus befejezését váltja ki mert a
# feltételt teljesíti,de az adatbázis aktuálisnak hagyva,
# rákérdez az új adatsor felvitelére.
i=$mezodb
;;

# Mégse gomb
3)

# A ciklus számlálót kettővel csökkenti.
# Ha csak eggyel csökkentené, akkor ugyanez a ciklus futna újra,
# mert a ciklus a számláló,eggyel való növelésével kezdődik.
# De nekünk az ezt megelőző kell.
i=$((i-2))
```

```
# Vissza állítja az előző ciklus, kezdő, adatait, amik a kettővel
# ezelötti ciklus befejező adatai.Ezeket teszi aktuálissá.
bevittadat=${bevittadatprev[$i]}
ujadatsor=${ujadatsorprev[$i]}
;;

# Az ablak be lett zárva. A scriptből kilépés
255)
    exit
;;
esac

done

if test ! $gomb -eq 1
then

# A bevitt adatok helyességét ellenőriztetjük a felhasználóval.
Xdialog --title "Shell Programozás 8/2 script" \
--backtitle "Helyessek az adatok ?" --default-no \
--yesno "$bevittadat\n" 0 0
v=$?

# Ha igennel válaszolt, akkor hozzáírja a fájlhoz.
# Ha "nem" gombot nyomott, vagy bezárta az ablakot, akkor ez
# kimarad, az adatokat eldobja.
if test $v -eq 0
then
    echo $ujadatsor >> $datfile
else
    Xdialog --title "Shell Programozás 8/2 script" \
    --msgbox "Az adatokat eldobtam ! " 0 0
fi

fi

# Ha nem-et válaszol, 1-re változtatja a $ki értékét,
# ami hatására a külső ciklus abbahagyja a futását.
Xdialog --title "Shell Programozás 8/2 script" \
--yesno "Kíván újabb adatsort felvenni ?" 0 0
ki=$?
# Ha az ablak be lett zárva, akkor 255,
# de a $ki akkorsem 0, vagyis a ciklus ugyanúgy befejeződik,
# mintha 1 lenne, azaz a "nem" gomb lett volna megnyomva.

done

exit
```

---

9. Scriptünk tesztelése, hibajavítása

Azt mondja néhány programozó, hogy a program megtervezése az összidő 1/3-ed része, a megírása másik 1/3-ed része, a hibakeresés és tesztelés pedig a maradék 1/3-ed rész.

Mit tehetünk, ha a scriptünk, egyszerűen nem akar működni és nem tudjuk mi a hiba.

Indítsuk a scriptet a -v vagy az -x kapcsolóval
```
$ bash -v scriptünk
$ bash -x scriptünk
```
Ezek futás közben értékes információkat adnak a számunkra.

Másik lehetőség, hogy a scripben echo-kat helyezünk el, amik informálnak minket.
```
if test
then
    echo "IF then ága fut"
else
    echo "IF else ága fut"
fi
```

Vagy közben, a különféle változók értékeit kiíratjuk és ezzel is vizsgáljuk mit is csinál a scriptünk.

```
echo "\$neve valtozó : $neve"
```

Esetleg ha futás közben a kiírás törlődne, például clear vagy más parancs miatt, akkor felfüggesztjük a futást, egy read-al. Fontos, hogy a read után olyan változót adjunk meg, amit egyébként nem használunk a scriptben.

```
echo "\$neve valtozó : $neve"; read tmp
```

Erre is használható az Xdialog:
```
Xdialog --msgbox "$neve" 0 0
```

10. Néhány ötlet

Néhány dolog nem igazán kívánkozott egyik részhez sem. Ezért itt felsorolom ezeket.
Különösebb magyarázat nélkül teszem közzé őket.

Néhány esetben szükségünk lehet véletlenszám generálásra:

```
$ tol=10
$ ig=15
$ int=$((ig+$tol))
veletlenszam=`echo $((RANDOM%$((intervallum+1))+$tol))`
```

Milyen hosszú egy adott string:

```
$ a="pirosalma napraforgó"
$ hossz`echo $a | wc -c | tr -d ' '`
$ echo $hossz
21
```

Hányszor fordul elő egy egysoros stringben, egy adott betű.

```
$ betu="a"
$ db=`echo $((`echo $a | tr "$betu" '\n' | wc -l | tr -d ' '`-1))`
$ echo $db
4
```

Hányszor fordul elő egy hosszabb kifejezés egy több soros szövegben.

```
$ kif="ter"
$ db=`cat adatok.dat | tr -d '\n' | sed s/"$kif"/'\n'/g | \
wc -l | tr -d ' '`
$ echo $db
4
```

Hogyan lehet egy karakter ascii kódját megtudni.

```
$ char="é"
$ kod=`echo $char | od -t u1 | grep 0000000 | cut -c 9,10,11 | sed
s/\ //g`
$ echo $kod
233
```

```
Egy ascii Kódot, hogyan lehet karakterré formálni.
$ kod=233
$ char=`echo -e "\\\\$(printf %03o $kod)"`
$ echo $char
é
```

Utószó

Befejezésképpen csak annyit szeretnék mondani, hogy örömömre szolgált, ha valakinek segíthettem s script írás elsajátításában, vagy a linuxhoz való közelebb kerüléséhez. Hiszen én már annyi segítséget kaptam, különféle fórumokon, levelezőlistákon, sőt személyesen is, hogy éppen ideje volt megpróbálni nekem is viszonozni belőle valamit.

Aki ezen a dokumentumon átverekedte magát, az ha eddig nem is foglalkozott programozással, nyugodtan neki állhat egy valódi programozási nyelvet megismerni. Bár itt csak a programozás legalapvetőbb néhány fogását ismerhette meg, de ez jó alapot nyújt ahhoz, hogy egy nagy lélegzetvétel után, belevesse magát a programozás rejtelmeibe. Én személy szerint a PHP-t, vagy a java nyelvet javasolom, mert platformfüggetlenek és mert elsősorban webes alkalmazások készítésére alkalmasak. Márpedig az internet fejlődése még messze nem ért a végére. Hazánk, pedig még csak most áll az általános felhasználók körébeni, "internet-robbanás" előtt, ami az EU csatlakozással nagyon rövid idő alatt be fog következni. A java már kilépett a webes fejlesztőeszközök köréből is. A jövő programnyelvének tartott, teljesen modern, objektum orientált nyelvé vált. De a megfelelő szerverek és programok telepítése után, a PHP is használható helyi gépen történő feladatmegoldásokra és a PHP4-től szintén nagyot lépett az objektumorientáltság felé, ami ma már alapvető elvárás egy modern nyelvtől. A PHP a shell után, mégis igen ismerősnek fog tűnni. Mind emellett, teljesen fel van készítve a különféle adatbázis kezelésekre is. Akár a Microsoftos vagy akár egy őskövületnek számító dbase, clipper esetleg szükség esetén szöveges alapú adatbázisokat is könnyedén tud kezelni.

Arra biztatok, minden érdeklődőt, hogy senki ne hagyja, hogy a kezdeti nehézségek letörjék az érdeklődését. Érdemes a linuxal foglalkozni, mert egyre többször és több helyen fogunk vele találkozni és egyre "felhasználóbarátabb" formában. Nem beszélve róla, hogy jelenleg az UHU Linux nevezhető a leginkább magyar operációs rendszernek. Mint linux disztribúció, pedig teljesen magyar fejlesztés.

Tisztelettel,
Raffai gábor István alias Glindorf

Xdialog jegyzet.

Xdialog 2.0.6

A leírást, Raffai Gábor István alias Glindorf készítette.

NÉV

Xdialog - grafikus dialógus ablakokat jelenít meg scriptekben. (A továbbiakban ablakok helyett, dialógus dobozokról beszélünk.)

SZINTAXIS

Xdialog [< általános opciók >] [< ideiglenes opciók >] < dialogusdoboz opciók > ...

LEÍRÁS

Az Xdialóg segítségével, könnyen kezelhetővé, felhasználóbaráttá és látványosabbá tehetjük scriptjeinket. Programozási ismeretek nélkül készíthetünk grafikus felületű segédprogramokat. Bármit, amit a shell lehetővé tesz. Az Xdialog, grafikus dialógus dobozokban kommunikál a felhasználóval. A shellből információkat közölhetünk és kérdéseket tehetünk fel vele a felhasználónak, majd a dialógus-doboz, a válaszokat át adja a scriptnek.

OPCIÓK

Az általános, az ideiglenes és a dialógus opciókkal határozhatjuk meg a doboz működését és adhatunk át a shellből a dialógus ablaknak. Minta "[ ]" zárójelek is mutatják, az általános és az ideiglenes opciók megadása nem kötelező, ekkor az alapértelmezés szerint történik a végrehajtás.

Általános opciók:

Az < általános opciók > -al, a dialógus-dobozok, (a következőkben csak "dobozok",) általános működését, stílusát és kinézetét szabhatjuk meg. Az itt beállított tulajdonságok, öröklődnek a következő dobozokra és csak a beállítás ellenkezőjének megadásával törölhetőek.

Lehetséges általános opciók:

--wmclass <név>

--rc-file <gtkrc fájlnév>

A kinézetet meghatározó más gtkrc fájl adható meg.

--backtitle <másodlagos felirat>

Ez a szöveg az ablakfejléc alá kerül. Ettől egy vonallal elválasztva kezdődik a valódid szöveg.

--title <ablakfelirat>

Az ablak fejléc-feliratát határozza meg.

--allow-close | --no-close

Bezárhatóvá ill. nem-bezárhatóvá teszi az ablakot. --no-close esetén, csak szabályosan, OK, Cancel, stb módon zárható be az ablak.

--screen-center | --under-mouse | --auto-placement

Az ablak a képernyő közepén, vagy az egérkurzor felett, illetve a automatikusan kiválasztott helyen jelenik meg.

--center | --right | --left | --fill

Az ablakban megjelenő szöveg igazítása állítható be.

--no-wrap | --wrap

--cr-wrap | --no-cr-wrap

--stderr | --stdout

Azt határozza meg, hogy a felhasználó válaszát, melyik kimenetre küldje. alapértelmezett a hiba-kimenet. A legtöbb esetben érdemes a normál kimenetre állítani (--stdout), mert az alapértelmezett a 2-es, hibakimenet.

--separator <karakter> | --separate-output

Mivel a dialógus doboz csak 1 soros formában adja visza a válaszokat, azon dobozok esetén, ahonnan több felhasználói adat érkezik a scripthez, beállítja, hogy a más-más mezők, milyen karakterrel legyenek elválasztva. Alapértelmezésben "/" jel. Ha azt szeretnénk, hogy sorokba rendezve kapjuk meg a válaszokat, akkor a "\n"-t kell beállítani. --separator ";" esetén pl. a következőképpen tehetjük külön-külön változóba a visszakapott értékeket:

```
valasz1resz=`echo "$valasz" | cut -d\; -f1`
valasz2resz=`echo "$valasz" | cut -d\; -f2`
valasz3resz=`echo "$valasz" | cut -d\; -f3`
```

Az alapértelmezett "/" szeparátor esetén és olyan szeparátoroknál,amiknek nincs különleges jelentősége a shell részére, cut-nál nem kell "\". Azaz elég:

```
valasz1resz=`echo "$valasz" | cut -d/ -f1`
```

--buttons-style default|icon|text.

A nyomógombok stílusát határozza meg.

Ideiglenes opciók:

The <transient options> only apply to the next <box option> into the Xdialog command line. These options are used to tune the widgets (number and type of buttons, menu icon) or to trigger some of the widgets specific features.

--fixed-font

Fixed font használata. --tailbox, --textbox és --editbox esetén.

--password

A dialógus doboz utolsó 1 vagy 2 mezőjét, jelszómezőnek állítja be. Ekkor a begépelés során ezekben csak "*" karakter látszik. alkalmas jelszó megadására, beállítására, megváltoztatására. --2inputsbox vagy --3inputsbox esetén.

--editable

Választólista esetén engedi az egyéni adatbeírást is, nem pedig csak a felkínált választhatóságokat. Csak --combobox esetén.

--time-stamp | --date-stamp

--logbox esetén az időbélyeget határozza meg.

--reverse

Fordított időrendben naplóz a --logbox -ban.

--keep-colors

A log színezése --logbox esetén.

--interval <timeout>

This option may be used with input(s) boxes, combo box, range(s) boxes, spin(s) boxes, list boxes, menu box, treeview, calendar and timebox widgets.

--no-tags

A tagok sorszámozását kapcsolja ki --menubox, --checklist és --radiolist dobozokban.

--item-help

A --menubox, --checklist, --radiolist, --buildlist és --treeview dobozok esetén használhatóak. Ekkor egy-egy tag esetében, nem csak 2 paraméter, (a tag és a szöveg,) hanem egy harmadik "help" is. Ez a doboz alsó részében megjelenő szöveg lesz, amely annak megfelelően változik, hogy éppen melyik tag van kijelölve.

--default-item <tag>

Beállítja a --menübox -ban az alapértelmezett választást.

--icon <xpm filename>

A beállított képet mint ikont, kihelyezi a megadott szöveg elé.

--no-ok

Megszünteti az OK gombot. --tailbox és --logox esetén.

--no-cancel

Megszünteti a Cancel, vagy Mégsem gombot. --infobox, --gauge és --progress esetén hatástalan.

--no-buttons

Minden gombot megszüntet. --textbox, --tailbox, --logbox, --infobox esetén a Help és a Print gombokat, --fselect és --dselect esetén pedig az "Új könyvtár", a "Fájl törlése" és a "Fájl átnevezése" gombok nem jelennek meg.

--default-no

Az alapértelmezett gomb, legyen a "Mégsem" vagy "Nem" gomb. --wizard opció használata esetén nem lehetséges.

--wizard

"Varázsló" stílusu módon jelenik meg három gomb. (Előző , Mégse , Következő.) --msgbox, --infobox, --gauge és --progress esetén hatástalan.

--help <help>

Egy súgó gombot jelenít meg a dobozon, amit ha megnyomunk, akkor egy msgbox-ban a --help paraméter után megadott szöveg jelenik meg. Majd OK után visszatér az eredeti doboz. --infobox, --gauge és --progress esetén hatástalan.

--print <printer>

--tailbox, --textbox és --editbox esetén határozható meg vele a nyomtató.

--check <címke>

Egy ellenkörző jelölőnégyzetet jelenít meg a dobozon. A doboz visszaadja, hogy kipipálta-e a felhasználó. A doboz által visszaadott értéktől egy "\n" azaz enterrel elválasztva kerül átadásra. De ha a válasz a scriptben egy változóba van irányítva, ott már csak egy szóköz jelenik meg köztük. Itt kétféle érték jelenhet meg, "checked" vagy "unchecked". --infobox, --gauge és --progress esetén hatástalan.

--ok-label <cimke>

Az OK gombon az itt megadott szöveg jelenik meg.

--cancel-label <címke>

Az Cancel/Mégsem és a No/Nem gombon az itt megadott szöveg jelenik meg. --wizard opció esetén hatástalan.

--beep

Csipog egyet, a doboz megjelenésekor.

--beep-after

Csipog egyet, a doboz becsukódásakor.

--begin <Yorg> <Xorg>

This option may be used with all widgets.

--ignore-eof

--infobox és --gauge esetén, nem vesz tudomást a fájlvégi EOF-ról. Ugyanis az megszakítaná a működését.

--smooth

--tailbox és --logbox esetén a teljes állományt beolvasva a memóriába, így símább a lapozást, scrollozást, de több menóriát is foglal le.

A dialógus doboz:

Ezek a dialógus doboz típusát és pozícióját, ill. méretét határozzák meg. Itt történik a fő paraméterek átadása is. A doboz által feldolgozandó karaktersorozat, vagy filenév.

A doboz pozíciójának, méreteinek meghatározása.

A karakter mérettel történő méret meghatározása:
< Szélesség > < Magasság > (pl. 10 20)
Képernyőpont mérettel történő meghatározás:
< Szélesség >x< Magasság > (pl. 160x300).

A <0> <0> esetén automatikusan, a doboz tartalma alapján történik a szükséges méret meghatározása. A <-1> <-1> esetén, a doboz, a teljes képernyőt kitöltve jelenik meg.

It is possible to give the widget an absolute origin on the screen (provided your window manager lets you do so), either by using the --begin transient option (when the size

is given in characters) or by using a -geometry"-like origin (e.g. 400x200+20-30).

Paraméterek átadása.

Sajnos azt vettem észre, hogy nem használható a felkiáltó jel "!". Ugyanis a bash azt különleges jelként értelmezi. Ez nem is lenne baj, hiszen a shell számára különleges jelentőségű karakterek ilyetén voltát meg lehet szüntetni, ha elé egy "\" jelet gépelünk. Sajnos a "\!" esetén viszont, a "\" jel is megjelenik a szövegben. :-(

A bevitt szövegben a sortörést a "\n" karakterek beírásával érhetjük el. "Ez a bevitt\nszöveg, amit\n megtördeltem." Ez így fog megjelenni:
Ez a bevitt
szöveg, amit
megtördeltem.

Három féle kimenetre kaphat választ a script a doboztól:

1. A hibakimenet: 2> vagy >&2
2. A normál kimenetre: > vagy 1> vagy >&1
3. A shell visszatérési változójában: $?

A választ általában, vagy egy fájlba, vagy egy változóba szeretnénk tenni. A következőképpen tehetjük meg ezeket:

1. Fájlba:
   Xdialog --inputbox "ez a kérdés" 0 0 > fájl
2. Változóba:
   1. valasz=` Xdialog --inputbox "ez a kérdés" 0 0 2>&1`
   2. valasz=` Xdialog --stdout --inputbox "ez a kérdés" 0 0`
3. A „samples” példascriptekben legtöbbször, a következő megoldást mutatják be:
   Xdialog --inputbox "ez a kérdés" 0 0 2> /tmp/ideiglenesfile.$$
   valasz=' cat /tmp/ideiglenesfile.$$'

Az arra vonatkozó adatot pedig, hogy a doboz mi módon lett bezárva, a $? program-visszatérési változóból tudhatjuk meg. Ennek értéke, 0, 1, 2, 3, vagy 255 lehet.
Jelentéseik:
0 : OK, Yes/Igen vagy Next/Tovább gomb megnyomása történt.
1 : Cancel/Mégsem vagy No/Nem gomb megnyomása történt.
2 : Help gomb megnyomása történt
3 : Previous/Előző gomb megnyomása történt. (csak --wizard opció esetén).
255 : Hibával zárult be, nem pedig normál módon. Ha a dialógus doboz,

"doboz-bezárás"-al let bezárva, akkor is ezzel az értékkel tér vissza.

Lehetséges dialógus dobozok és a megadható paraméterek:

--yesno <karaktersor> <magasság> <szélesség>

Megjeleníti a szöveget és egy "Igen" meg egy "Nem" gombot. A válasz a $? változóból tudható meg:

```
Xdialog --yesno "Indulhat az akció ?" 0 0
valasz=$?

if test $valasz = 0; then
  <igent nyomott>
fi

if test $valasz = 1; then
  <nemet nyomott>
fi

if test $valasz = 255; then
  <bezárta az ablakot>
fi
```

--msgbox <karaktersor> <magasság> <szélesség>

Egy üzenetet jelenít meg, egy "OK" gombbal.

```
Xdialog --title "Üzenet !!!" --msgbox "Üzenem hogy,... ?" 0 0
```

--infobox <karaktersor> <magasság> <szélesség> [<időmeghatározás>]

Szintén egy üzenetet jelenít meg, de meghatározható, hogy a doboz egy adott idő elteltével, magától bezáródjon. Az időmeghatározás egy szám, mely ezredmásodpercben határozza meg az időt. Amint a "[ ]" zárójel is jelzi, megadása nem kötelező. Ha nem adunk meg semmit, akkor az alapértelmezett 1000 , azaz 1 mp lép érvénybe.

--gauge <karaktersor> <magasság> <szélesség> [<százalékszám>]

Egy folyamat előrehaladását ábrázolja mértéksávon jelezve. A grafikus sáv jelzi a százalékot, ahol a folyamat tart. A folyamat előrehaladottságának százalékértékekét a szabványos bemenetről kapja meg. Amennyiben újabb százalékértéket kap, frissíti a sávot, így jelezi az új százalékértéket. Ha a bemeneten az "XXX" karaktersort kapja meg, akkor az ez után a szabványos bemenetről jövő karaktereket, a sáv felett, mint szöveget jeleníti meg, felül írva az előző, itt helyet foglaló szöveget. Ez a következő "XXX" karaktersorig történik. Onnan újra mint százalékértékként értelmezi a bemeneten kapott számot. A bemeneten érkező első EOF jelre befejeződik a működése. Ezt az --ignore-eof kapcsolóval lehet elkerülni. Ha a folyamat túl gyors lenne és szeretnénk követhetővé tenni,használjuk közben a sleep <mp> parancsot, ahol a megadott másodpercre felfüggeszti a shell a futást.

```
(
 sleep 3
 echo "25"
 echo "XXX"; echo "huszonöt százalék"; echo "XXX"
 sleep 3
 echo "50"
 echo "XXX"; echo "ötven százalék"; echo "XXX"
 sleep 3
 echo "75"
 echo "XXX"; echo "hetvenöt százalék"; echo "XXX"
 sleep 3
 echo "100"
 echo "XXX"; echo "száz százalék"; echo "XXX"
 sleep 3
 ) | Xdialog --gauge "nulla százalék" 0 0
```

--progress <karaktersor> <magasság> <szélesség> [<maxdots> [[-]<msglen>]]

Ez szintén egy folyamat előrehaladását jelzi ki mértéksávon. De nem kell, hogy a százalékértékeket szám formájában megkapja, hanem a bemenetére helyezett folyamatot jelzi ki.

```
find "$HOME" *.mp3 | Xdialog --progress "mp3 keresés" 0 0
```

--inputbox <karaktersor> <magasság> <szélesség> [<alapérték>]

valasz=` Xdialog --stdout --title "Adatbevitel" --backtitle "Név bevitel." --inputbox "Írd be a teljes nevedet." 0 0 "Mr. "`

Egy egymezős adatbeviteli dobozt jelenít meg. A <karaktersor> lesz a kérdés. Az <alapérték> jelenik meg a beviteli mezőben.

--2inputsbox <karaktersor> <magasság> <szélesség> <cimke1> <alapérték1> <cimke2> <alapérték2>

Két darab beviteli mezőt jelenít meg. Mivel itt elvileg két értékkel tér vissza a doboz, (kereszt és vezetéknév,) csakhogy a visszatérés egyetlen sorban történik, ekkor alapértelmezésben egy "/" jel válassza el a két értéket. Pl Kovács Gábor esetében a $valasz változóba ez kerül: "Kovács/Gábor" A "/" helyett más szeparátor jel is meghatározható a --separatot <x> kapcsolóval. pl --separator ";" vagy a --separator "\n" esetén egy enterrel választja el a két mező értékét. Felhasználható jelszó bekérésre is.

valasz=` Xdialog --stdout --separator ";" --title "Adatbevitel" --backtitle "Név bevitel." --2inputsbox "Írd be a teljes nevedet." 0 0 "Keresztnév:" "" "Vezetéknév:" "Mr. "`

--3inputsbox <karaktersor> <magasság> <szélesség> <cimke1> <alapérték1> <cimke2> <alapérték2> <cimke3> <alapérték3>

Ugyanaz mint a --2inputsbox, de 3 mezővel. Használható pl. jelszó módosításra, vagy új jelszó kérésre. Mint a --2inputsbox esetében, itt is beállítható, hogy mely mezőkbe történő adatbevitelnél, legyenek láthatók, csak "*" karakterek. Ezt a --password kapcsolóval lehet elérni. Egyszeri beírásakor, az utolsó mező, kétszeres beírásakor az utolsó két mező, háromszori esetén mindhárom mezőben csak "*" karakterek lesznek láthatóak. A --password esetén, alul látható egy jelölő négyzet, "A gépelt szöveg elrejtése" felirattal. Ha ez elöl kivesszük a pipát, akkor "*" helyett, láthatóvá válnak a betűk.

valasz=` Xdialog --stdout --separator ";" --password --password --title "Beállítások" --backtitle "Jelszóbeállítás" --3inputsbox "Kérem adja meg a kívánt jelszót" 0 0 "User név:" "" "Jelszó:" "" "Jelszó megerősítése:" ""`

valasz=` Xdialog --stdout --separator ";" --password --password --password --title "Beállítások" --backtitle "Jelszó módosítás" --3inputsbox "Kérem módosítsa a jelszavát" 0 0 "Régi jelszó:" "" "Új jelszó:" "" "Az új jelszó megerősítése:" ""`

--combobox <karaktersor> <magasság> <szélesség> <választás1> <választás2> ... <választásN>

Lenyíló választólista mezőt hoz létre. A <választásN> -ben megadott szövegek lesznek kiválaszthatóak. A kiválasztott szöveggel tér vissza a scripthez, nem pedig annak sorszámával.

valasz=` Xdialog --stdout --combobox "Válasszon ki egy elemet." 0 0 "Első tag" "Második tag" "Harmadik tag"`

--rangebox <karaktersor> <magasság> <szélesség> <minimum érték> <maximum érték> [<alapértelmezett érték>]

Ez egy csúszkát jelenít meg, amelyen egy numerikus érték adható meg. Megadható neki a legkisebb, a legnagyobb és az alapértelmezett érték.

valasz=` Xdialog --stdout --rangebox "Hány éves ön ?" 0 0 "18" "90" "30"`

--2rangesbox <karaktersor> <magasság> <szélesség> <cimke1> <minimum1> <maximum1> <alapé1> <cimke2> <minimum2> <maximum2> <alapé2>

Mint a --rangebox esetében, de két csúszkát jelenít meg egy dobozban. Mindegyikhez külön felirat, maximum és minimum, valamint alapérték rendelhető, a scripthez a szokásos --separator -al elválasztva egy sorban tér vissza az érték.

--3rangesbox <karaktersor> <magasság> <szélesség> <cimke1> <minimum1> <maximum1> <alapé1> <cimke2> <minimum2> <maximum2> <alapé2> <cimke3> <minimum3> <maximum3> <alapé3>

Mint a --rangebox esetében, de három csúszkát jelenít meg egy dobozban. Mindegyikhez külön felirat, maximum és minimum, valamint alapérték rendelhető, a scripthez a szokásos --separator -al elválasztva egy sorban tér vissza az érték.

--spinbox <karaktersor> <magasság> <szélesség> <minimum> <maximum> <alapé> <címke>

Mint a --rangebox -nál, de itt az egyesnél is megadható címke. Itt viszont nem egy csúszkán, hanem egy kicsi értékbeállító spin-el adható az meg. Itt nagyobb érték is beírható mint a maximumban meghatározott, viszont a scriptnek akkor is csak a maximumban meghatározott értéket adja vissza.

--2spinsbox <karaktersor> <magasság> <szélesség> <minimum1> <maximum1> <alapé1> <cimke1> <minimum2> <maximum2> <alapé2> <cimke2>

Mint a --rangebox, de két spin dobozzal.

--3spinsbox <karaktersor> <magasság> <szélesség> <minimum1> <maximum1> <alapé1> <cimke1> <minimum2> <maximum2> <alapé2> <cimke2> <minimum3> <maximum3> <alapé3> <cimke3>

Mint a --rangebox, de három spin dobozzal.

valasz=` Xdialog --stdout --spinbox "Kérem módosítsa a jelszavát" 0 0 "1900" "2003" "1970" "Születési év" "1" "12" "6" "A születés hónapja" "1" "31" "15" "A születés napja"`

--textbox <file> <magasság> <szélesség>

Egy szöveges fájl tartalma jeleníthető meg vele. Egy ideiglenes fájlon keresztül a szabványos kimenet is megjeleníthető vele. A --no-buttons kapcsoló nem használható nála. Ha a fájlnév helyett a "-" adjuk meg, akkor a szabványos bemenetről olvas, aza pipeline-al más programoktól, parancsok szabványos kimenetéről tudja átvenni a megjelenítendő szöveget. (Lásd.: --editbox példa.)

```
mplayer --help > /tmp/boxtmp.$$
Xdialog --textbox "/tmp/boxtmp.$$" 0 0
```

--editbox <file> <magasság> <szélesség>

Hasonló mint a textbox, de szerkeszteni is lehet a fájlt vele. az alábbi módon a felhasználó általi változtatások elmentésre kerülnek a fájlba.

```
cat file.txt | Xdialog --stdout --editbox "-" 0 0 > file.txt
```

--tailbox <file> <magasság> <szélesség>

Majdnem ugyan olyan mint a --textbox de jelentősen gyorsabb scrollozást, megjelenítést tesz lehetővé. Ugyanakkor itt már használható a --no-buttons kapcsoló is. A "-" fájlnévként való megadásával ugyanúgy képes a szabványos bemenetről olvasni (pipeline). --textbox helyett inkább ez tűnik jobb választásnak.

--logbox <file> <magasság> <szélesség>

A logbox leginkább annyiban különbözik a tailbox-tól, hogy a --time-stamp kapcsolóval idő bejegyzést szúrhatunk minden sor elé, a --date-stamp -al pedig a dátumot is. Illetve a --keep-colors kapcsolóval színezhető a megjelenítés.

--menubox <karaktersor> <magasság> <szélesség> <menü magasság> <azonosito1> <menupont1> {<help1>}

Egy függőleges menüt jelenít meg. A menü magasság egy szám ami azt jelenti, hogy egyszerre csak az itt megadott számú menüpont látszik, a többit görgetéssel lehet elérni. Egy menüponthoz két adat adható meg, de ha az --item-help kapcsoló használva van, akkor egy harmadik is. De csak akkor. (Ezt jelöli a {} zárójel) A három adat:

1. a menü azonosítója. (Ésszerűnek tűnik itt sorszámokat használni.) Ezt az értéket fogja az ablak átadni a scriptnek.
2. A menüpont szövege.
3. A menüpont help-je, vagy többletinformáció róla. Ez a doboz alsó részében, de a gombok felett jelenik meg. A --no-tags kapcsolóval, elrejthető a menüpontok azonosítói.

valasz=` Xdialog --no-tags --item-help --menubox "Válasszon a menüből." 0 0 3 "1" "Első pont" "első pont segítsége" "2" "Második pont" "második pont segítsége" "3" "harmadik pont" "harmadik pont segítsége" "4" "negyedik pont" "negyedik pont segítsége" "5" "ötödik" "ötödik pont segítsége"`

--checklist <karaktersor> <magasság> <szélesség> <lista magasság> <azonosító1> <elem szöveg1> <státusz1> {<help1>}

Egy jelölőnégyzet listát ad. Itt a szöveg előtti rész pipálható ki, vagy vehető ki a pipa előle. A működése hasonló mint a --menubox -é, annyi különbséggel, hogy mivel itt több elem is kiválasztható, kipipálható, ezért a doboz a visszatérési értékben a --separator és a --2inputsbox -nál ismertetett módon adja vissza a kijelölt elemek azonosítójának, a szeparátor karakterrel elválasztott listáját. A --menubox -hoz képest, viszont listaelemenként bővül eggyel a megadandó paraméterek száma.

1. jelölőnégyzet listaelem azonosítója.
2. Az elem szövege.
3. Az elem státusza (ez a plusz)
4. A jelölőnégyzet elem helpje. (Csak --item-help esetén.)

A 3. paraméter, az elem státuszánál, háromféle érték adható meg.

1. "off": az elem a doboz megnyílásakor nincs kijelölve.
2. "on": az elem a doboz megnyílásakor kivan jelölve.
3. "unavailable" Ez azt jelenti, hogy az elem megjelenik, de nem-kiválasztható.

(--item-help esetén a help szövegeket nem a doboz alján, hanem ha az egérrel 0.5mp-re megállunk az elem felett, akkor jeleníti meg, egy kis címkén az egér mellett.)

valasz=` Xdialog --stdout --no-tags --item-help --checklist "Pipálja ki a megfelelőket." 0 0 3 "1" "Első pont" "off" "első pont segítsége" "2" "Második pont" "on" "második pont segítsége" "3" "harmadik pont" "on" "harmadik pont segítsége" "4" "negyedik pont" "unavailable" "negyedikpont segítsége" "5" "ötödik" "off" "ötödik pont segítsége"`

--radiolist <karaktersor> <magasság> <szélesség> <lista magasság> <azonosító1> <elem1> <státusz1> {<help1>}

A rádiógomb listát ad ki. Minden úgy működik mint a --checklist esetében, azzal a különbséggel, hogy mivel itt egyszerre csak egy elem jelölhető meg, ezért a státusz megadásakor is csak egy lehet "on". (Az --item-help itt is egér melletti címke.)

--buildlist <karaktersor> <magasság> <szélesség> <lista magasság> <azonosító1> <elem1> <státusz1> {<help1>}

Ez egy olyan dobozt jelenít meg, amiben két lista látható és az egyik listából a másikba helyezhetünk át elemeket. Végül is egy listából lehet több elemet kiválasztani vele. Az "on" státuszúak, rögtön a kiválasztottak között vannak. A doboz a kiválasztottak listájában lévő elemek azonosítóit adja vissza, a szeparátorral elválasztva.(Az --item-help itt is egér melletti címke.)

--treeview <karaktersor> <magasság> <szélesség> <lista magasság> <azonosító1> <elem1> <státusz1> <elem1 mélység> {<help1>}

Ez egy fastruktúra szerűen jeleníti meg az elemeket, amik így egymásba ágyazhatóak. Ezért egy ötödik a paraméter lép be, az elem mélység. Itt oda kell figyelni, hogy az elemek és a mélység értékek, helyesen kövessék egymást. Úgy kell a listának olyan sorrendben kell következnie, mintha minden szál kibontott állapotban lenne a fastruktúrában.

--fselect <file> <magasság> <szélesség>

Ez egy jól ismert fájl kiválasztó doboz. A file paraméter, az egy szűrő. Csak az ennek a szűrőnek megfelelő fájlokat fogja mutatni. Ha azt akarjuk, hogy mindent mutasson, "*"-ot kell neki megadni. Lehetőség van könyvtár létrehozására, fájl törlésére és fájl átnevezésére is. A --no-buttons kapcsoló, csak ezeket a lehetőségeket veszi el. A --check is használható. A fájl teljes elérési útjával és nevével tér vissza a scripthez. A --check egy "\n", azaz enterrel elválasztva a fájlnévtől, kerül a kimenetre.

fajl=` Xdialog --help "legyél ügyes és válassz ki egy fájlt." --check "Meg győződtél a helyes kiválasztásról ?" --fselect "*.gz" 0 0`

--dselect <directory> <magasság> <szélesség>

Mint az --fselect, de könyvtár jelölhető ki vele.

--calendar <karaktersor> <magasság> <szélesség> <nap> <hónap> <év>

Egy naptárat jelenít meg. Megadható, hogy milyen dátummal induljon. Az évnek minimum 1970-nek kell lennie. "nap/hónap/év" formában adja vissza a beállított dátumot.

--timebox <karaktersor> <magasság> <szélesség>

Egy időpont bevitelére alkalmas doboz. Három spin elem jelenik meg rajta, óra, perc, másodperc. Alapértéke az aktuális idő. Visszatérni, "óra:perc:mp" formában tér vissza.

Példa scriptek a következő elérési úton találhatóak:
/usr/local/share/doc/Xdialog-2.0.6/samples/*

MAN angol:
/usr/local/man/man1/Xdialog.1*

Dokumentumok angol nyelven:
/usr/local/share/doc/Xdialog-2.0.6/*
