Bevezetés a bash programozásba HOGYAN

Mike G mikkey at dynamo.com.ar 2000. július 27.

Ezen dokumentum célja, hogy bevezessen az alap- illetve középszintű shell szkript írásba. A dokumentum nem egy minden részletre kiterjedő leírás (ez a címéből is kitűnik). Én nem vagyok se profi shell programozó, se guru. Azért döntöttem úgy, hogy megírom ezt a HOGYANt, mert sokat tanulhatok belőle és talán másoknak is hasznára válhat. Minden visszajelzésért hálás vagyok, különösképpen, ha folt (patch) formájában küldöd el. :)

Contents

1 Bevezetés

1.1 A dokumentum legújabb változata

A dokumentum legújabb változata megtalálható a

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html <http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

honlapon

1.2 Szükséges alapok

Jól jöhet, ha ismered a GNU/Linux rendszerek parancssorát, és hasznos, ha tisztában vagy az alapvető programozási elvekkel. Noha ez nem egy programozásba bevezető leírás, sok ilyet is megpróbálok majd bemutatni.

1.3 A dokumentum használata

Ez a dokumentum a következő esetekben lehet hasznos:

- Már tudsz valamennyire programozni, és szeretnél végre shell szkripteket írni.
- Van valami halvány fogalmad a shell programozásról, és szeretnél többet megtudni róla.
- Látni akarsz pár shell szkriptet és némi magyarázatot, hogy hozzáláthass sajátod megírásához.
- DOS/Windows rendszerről jössz/jöttél és "batch" programokat akarsz írni.
- Annyira pedál vagy, hogy már minden létező HOGYANt elolvastál.

1.4 Magyar fordítás

A magyar fordítást *Kovács Ferenc* <mailto:Kovacs.Ferenc.5[kukac]stud.u-szeged[pont]hu> készítette (2004.06.18). A lektorálást *Daczi László* <mailto:dacas[kukac]freemail[pont]hu> végezte el (2004.06.20). A dokumentum legfrissebb változata megtalálható a *Magyar Linux Dokumentációs Projekt* <http://tldp.fsf.hu/> honlapján. A dokumentum fordítása a *Szegedi Tudományegyetem* <http://www.u-szeged.hu/>

nyílt forráskódú szoftverfejlesztés speciálkollégium <http://www.inf.u-szeged.hu/~havasi/opensource/> ának segítségével valósult meg.

2 Egyszerű szkriptek

Ez a HOGYAN megpróbál útmutatóként szolgálni a shell szkriptek megismeréséhez. A témát gyakorlati oldalról, példákon keresztül közelítjük meg.

Az ebben a részben található néhány példa remélhetőleg segít majd megérteni pár alapvető módszert.

2.1 A szokásos hello world szkript

```
#!/bin/bash
echo Hello World
```

A szkript csak két sorból áll. Az elsőben jelezzük a rendszernek, hogy melyik programot használja a szkript futtatásához.

A másodikban pedig kiírjuk a "Hello World" szöveget a terminálra.

Ha a képernyőn a *./hello.sh: Command not found.* vagy ehhez hasonló üzenet jelenik meg, akkor valószínűleg az első sorban lévő "#!/bin/bash" lesz a ludas. A hiba javításához módosítsd az első sort úgy, hogy az a bash tényleges elérési útvonalát tartalmazza. Az útvonal lekérésére használható a "whereis bash" parancs, vagy olvasd el "A bash felkutatása" című fejezetet.

2.2 Egy nagyon egyszerű backup szkript

```
#!/bin/bash
tar -cZf /var/my-backup.tgz /home/me/
```

Ebben a szkriptben, az előzőtől eltérően, nem szöveget jelenítünk meg a képernyőn, hanem a felhasználó "home" könyvtárában lévő fájlokat összefűzzük egyetlen tar fájlba. Ez csak a példa kedvéért szerepel most itt, később egy sokkal használhatóbb backup szkriptet is bemutatunk.

3 Az átirányításról

3.1 Elmélet és gyors útbaigazítás

Három fájlleíró van: stdin, stdout és stderr (std=standard).

Alapvetően a következők tehetők meg:

1. stdout fájlba irányítása
2. stderr fájlba irányítása
3. stdout stderr-be irányítása
4. stderr stdout-ba irányítása
5. stderr és stdout fájlba irányítása
6. stderr és stdout stdout-ba irányítása
7. stderr és stdout stderr-be irányítása

Az 1 az stdout, a 2 az stderr (a 0 pedig az stdin - a ford.).

Egy kis magyarázat, hogyan is kell mindezt érteni: a less paranccsal meg tudod nézni mind az stdout "tartalmát" (ami a bufferen marad), mind az stderr-ét, ami a képernyőn jelenik meg, de törlődik, amint olvasni próbálod a buffert.

3.2 Példa: az stdout fájlba irányítása

Ennek hatására a program kimenete fájlba íródik.

```
ls -l > ls-l.txt
```

Itt létrejön az "ls-l.txt" nevű fájl, melynek tartalma meg fog egyezni azzal, amit az "ls -l" parancs futtatásakor a képernyőn látnál.

3.3 Példa: az stderr fájlba irányítása

Hatására a program stderr kimenete fájlba íródik.

```
grep da * 2> grep-errors.txt
```

Itt is létrejön a "grep-errors.txt" nevű fájl, melynek tartalma a "grep da *" parancs stderr kimenetével fog megegyezni.

3.4 Példa: az stdout stderr-be történő irányítása

Ennek hatására a program stderr kimenetére lesznek írva az stdout-ra küldött adatok is.

```
grep da * 1>&2
```

Itt a parancs stdout kimenete az stderr-re lesz elküldve, ahogy ezt már bizonyára észrevetted.

3.5 Példa: az stderr stdout-ba történő irányítása

Most a program stderr kimenete az stdout-nak megfelelő fájlleíróba lesz írva.

```
grep * 2>&1
```

Itt az stderr-ra küldött adatok is az stdout-on jelennek meg. Ha ezt egy csővel (pipe) összekötöd a less programmal láthatod, hogy azok a sorok, amik normális esetben eltűnnének (mivel az stderr-ra írtuk őket), most megmaradnak (hiszen az stdout-ra lettek irányítva).

3.6 Példa: az stderr és stdout fájlba irányítása

Ebben az esetben a program minden kimenete fájlba kerül. Ez néha hasznos lehet például a cron bejegyzések esetén, ha egy programot teljes csöndben akarsz futtatni.

```
rm -f $(find / -name core) &> /dev/null
```

Ez (gondolva itt a cron bejegyzésekre) minden könyvtárból kitörli a "core" nevű fájlokat. Fontos tisztában lenned azzal, hogy egy program pontosan mit is csinál, mielőtt "megszünteted" a kimenetét.

4 Csövek (pipe)

Ebben a fejezetben a csövek használatának egy egyszerű és hasznos módját ismertetjük, valamint elmondjuk hol lehet szükséged rájuk.

4.1 A csövek és azok használata

A csövek olyan egyszerű eszközök, melyek lehetővé teszik, hogy egy program bemenetét egy másik program kimenetéről vegye.

4.2 Példa: csövek és a sed

Ez a csövek használatának egy nagyon egyszerű módja.

```
ls -l | sed -e "s/[aeio]/u/g"
```

Itt a következő történik: először az ls -l parancs végrehajtódik, de a kimenete nem lesz kiírva a képernyőre, hanem el lesz küldve a sed programnak, ami ezt bemenetként értelmezi és ebből előállítja saját kimenetét.

4.3 Példa: az ls -l *.txt parancs másként

A következő parancs minden bizonnyal sokkal bonyolultabb, mint a szokásos ls -l *.txt, de a csövek tulajdonságainak illusztrálására jó példa. A gyakorlatban ilyen listázási problémák megoldására lehetőleg ne használjunk csöveket.

```
ls -l | grep "\.txt$"
```

Itt az ls -l program kimenete el lesz küldve a grep programnak, ami kiírja azokat a sorokat, melyek illeszkednek a "\.txt$" reguláris kifejezésre.

5 Változók

Ugyanúgy használhatsz változókat, mint bármely más programozási nyelvben. Adattípusok nincsenek. Egy bash változó tárolhat számot, karaktert és karakterekből álló sztringet.

A változókat nem kell külön deklarálni, a rájuk való hivatkozáskor automatikusan létrejönnek.

5.1 Példa: Hello World! változókkal

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

A második sorban létrejön egy STR nevű változó, melynek értéke a "Hello World!" (sztring) lesz. Ezután a változó neve elé tett "$" jellel tudjuk elérni annak értékét. Jegyezd meg (és próbáld is ki), hogy ha nem használod a "$" jelet, a program kimenete más lesz és valószínűleg nem az, amit szeretnél.

5.2 Példa: Egyszerű backup szkript (kicsivel jobb változat)

```
#!/bin/bash
OF=/var/my-backup-$(date +%Y%m%d).tgz
tar -cZf $OF /home/me/
```

Ez a szkript felvet egy másik dolgot is. De mindenek előtt tisztában kell lenned a változók létrehozásával, az értékadással kapcsolatos dolgokkal (ez a második sorban látható). Most vegyük szemügyre a "$(date +%Y%m%d)" kifejezést. Ha futtatod a szkriptet a zárójelek közti parancs is lefut, létrehozva kimenetét.

Azt is vegyük észre, hogy a kimeneti fájl neve minden nap más és más lesz, a date parancs formátum-kapcsolóinak (+%Y%m%d) köszönhetően. Ezt módosíthatod más formátum megadásával.

Még néhány példa:

echo ls

echo $(ls)

5.3 Lokális változók

Lokális változókat a *local* kulcsszóval tudunk létrehozni.

```
#!/bin/bash
HELLO=Hello
function hello {
        local HELLO=World
        echo $HELLO
}
echo $HELLO
hello
```

```
echo $HELLO
```

Ennek a példának elégnek kell lennie, hogy bemutassa a lokális változók használatát.

6 Feltételes utasítások

A feltételes utasítások segítségével dönthetünk, hogy végrehajtunk-e valamit vagy nem. A döntés itt egy kifejezés kiértékeléséből áll.

6.1 Elmélet

A feltételes utasításoknak több formája is van. A legalapvetőbb a következő: **if** *kifejezés* **then** *utasítás*, ahol az "utasítás" csak akkor kerül végrehajtásra, ha a kifejezés, kiértékelése után igazat ad vissza. Például a "2<1" kifejezés kiértékelés utáni értéke hamis, míg a "2>1" esetén igaz.

A másik eset: **if** *kifejezés* **then** *utasítás1* **else** *utasítás2*. Az "utasítás1" akkor hajtódik végre, ha a kifejezés igaz, egyéb esetben az "utasítás2" kerül végrehajtásra.

A feltételes utasítás egy másik formája: **if** *kifejezés1* **then** *utasítás1* **else if** *kifejezés2* **then** *utasítás2* **else** *utasítás3*. Ebben az esetben csak az "ELSE IF 'kifejezés2' THEN 'utasítás2'" részt adtuk hozzá, így az utasítás2 akkor lesz végrehajtva, ha a kifejezés2 kiértékelés utáni értéke igaz. A többit már magad is el tudod képzelni (nézd meg az előző formákat).

Pár szó a szintaxisról:

Bash-ban az "if" szerkezetek a következőképp néznek ki:

if [kifejezés];

then

a "kifejezés" igaz volta esetén végrehajtódó kód

fi

6.2 Példa: alapvető feltételes utasítás (if ... then)

```
#!/bin/bash
if [ "foo" = "foo" ]; then
   echo a feltétel igaz
fi
```

A szögletes zárójelbe tett kifejezés igaz értéke esetén végrehajtódó kód a "then" kulcsszó után és a "fi" kulcsszó előtt helyezkedik el. A "fi" kulcsszó jelzi a feltételesen végrehajtandó kód végét.

6.3 Példa: alapvető feltételes utasítás (if ... then ... else)

```
#!/bin/bash
if [ "foo" = "foo" ]; then
   echo a feltétel igaz
else
   echo a feltétel hamis
```

```
fi
```

6.4 Példa: Változók és feltételes utasítások

```
#!/bin/bash
T1="foo"
T2="bar"
if [ "$T1" = "$T2" ]; then
    echo a feltétel igaz
else
    echo a feltétel hamis
fi
```

7 Ciklusok (for, while és until)

Ebben a részben a for, while és until ciklusokról lesz szó.

A **for** ciklus egy kicsit eltér a más nyelvekben megszokottaktól. Alapvetően egy sztringekben szereplő szavak halmazának feldolgozására használjuk.

A **while** ciklus addig ismétli a ciklusmagot, amíg a ciklusfeltétel hamissá nem válik (vagy ki nem ugrunk a ciklusmagból a break utasítással).

Az **until** ciklus tulajdonképpen azonos a while ciklussal, a különbség csupán annyi, hogy az ismétlés addig folytatódik, amíg a feltétel hamis.

Ha gyanús volt számodra, hogy a while és az until ciklusok mennyire hasonlók, akkor igazad volt.

7.1 Példa for ciklusra

```
#!/bin/bash
for i in $( ls ); do
    echo elem: $i
done
```

A második sorban deklaráltunk egy ”i” nevű változót, ami majd sorra felveszi a $(ls) sztring különböző értékeit.

A harmadik sor szükség esetén méretesebb is lehet, vagy több sor is szerepelhet a ciklusmagban a done (4. sor) kulcsszó előtt.

A ”done” (4. sor) jelzi, hogy a $i értéket használó kódrészlet befejeződött. Innentől kezdve a $i új értéket vehet fel.

Ennek a szkriptnek így nem sok értelme van. A for ciklus használatának egy sokkal hasznosabb módja, ha az előző példánál csak bizonyos fájlokat vizsgálunk.

7.2 C-szerű for ciklus

Fiesh tanácsára került ide ez a példa, ami egy C/Perl-szerű for ciklust mutat be.

```
#!/bin/bash
for i in ‘seq 1 10‘;
do
        echo $i
done
```

7.3 While példa

```
#!/bin/bash
COUNTER=0
while [  $COUNTER -lt 10 ]; do
    echo A számláló értéke: $COUNTER
    let COUNTER=COUNTER+1
done
```

Ez a szkript a jól ismert ”for” ciklust utánozza (C, Pascal, perl, stb.).

7.4 Until példa

```
#!/bin/bash
COUNTER=20
until [  $COUNTER -lt 10 ]; do
    echo COUNTER $COUNTER
    let COUNTER-=1
done
```

8 Függvények

Mint a legtöbb programozási nyelvben itt is használhatunk függvényeket a kód logikailag összetartozó részeinek csoportosítására, vagy ha csak áldozni szeretnénk a rekurzió virtuális oltára előtt (azaz ha rekurzív függvényeket akarunk írni).

A függvénydeklaráció csupán ennyiből áll: function my_func { my_code }.

A függvény hívása pedig ugyanúgy történik, mint bármelyik program esetén, csak leírjuk a nevét.

8.1 Példa függvényekre

```
#!/bin/bash
function quit {
    exit
}
function hello {
    echo Hello!
}
hello
quit
echo foo
```

A 2-4 sorok tartalmazzák a "quit", az 5-7 sorok pedig a "hello" függvényt. Ha nem teljesen világos számodra, hogy mit is csinál a szkript, próbáld ki!

Megjegyzendő, hogy a függvénydeklaráció sorrendje tetszőleges.

Amikor lefuttatjuk a szkriptet látjuk, hogy először a "hello", majd a "quit" függvények hívódnak meg és a program sohasem éri el a 10. sort (echo foo).

8.2 Példa paraméteres függvényekre

```
#!/bin/bash
function quit {
   exit
}
function e {
    echo $1
}
e Hello
e World
quit
echo foo
```

Ez a szkript nagyrészt megegyezik az előzővel. A fő különbség az "e" nevű függvényben rejlik, ami nem csinál mást, mint kiírja az első átvett argumentumot. A függvényeknél szereplő argumentumok ugyanúgy kezelhetők, mint a szkriptnek átadott argumentumok.

9 Felhasználói felületek

9.1 Egyszerű menü a "select" használatával

```
#!/bin/bash
OPTIONS="Hello Quit"
select opt in $OPTIONS; do
    if [ "$opt" = "Quit" ]; then
     echo done
     exit
    elif [ "$opt" = "Hello" ]; then
     echo Hello World
    else
     clear
     echo rossz válasz
    fi
done
```

Ha lefuttatod a szkriptet láthatod, hogy ez a programozók álma, ha szöveges menükről van szó. Biztos észrevetted, hogy ez nagyon hasonlít a "for" ciklusra. Csak itt ahelyett, hogy minden $OPTIONS elemre egyszerűen végrehajtanánk a ciklust, mindig "megállítjuk" a felhasználót.

9.2 A parancssor használata

```
#!/bin/bash
if [ -z "$1" ]; then
    echo használat: $0 könyvtárnév
    exit
fi
SRCD=$1
TGTD="/var/backups/"
OF=home-$(date +%Y%m%d).tgz
tar -cZf $TGTD$OF $SRCD
```

Világosnak kell lennie számodra annak, hogy ez a szkript mit csinál. Az első feltételben szereplő kifejezéssel megnézzük, hogy a program kapott-e argumentumot ($1). Ha nem, kilépünk, és egy használati útmutatót jelenítünk meg a felhasználónak. A program többi részének ezen a ponton már tisztának kell lennie.

10 Egyebek

10.1 Adatok bekérése felhasználótól: read

Sok esetben szükség lehet arra, hogy adatokat kérjünk be a felhasználótól. Erre számos lehetőségünk van, melyek közül egy a következő:

```
#!/bin/bash
echo Kérlek add meg a neved
read NAME
echo "Hi $NAME!"
```

A read segítségével egyszerre több adatot is be tudsz olvasni. A következő példa ezt tisztázza.

```
#!/bin/bash
echo Kérlek add meg a vezeték- és keresztneved
read FN LN
echo "Hi! $LN, $FN !"
```

10.2 Aritmetikai kifejezések kiértékelése

A parancssorba (vagy a shell-be) írd be a következőt:

echo 1 + 1

Ha ”2”-t vártál eredményül bizonyára csalódott vagy. De mi van akkor, ha azt szeretnéd, hogy a bash ki is értékelje a számokat? A megoldás a következő:

echo $((1+1))

Ez már sokkal logikusabb kimenetet ad. Így kell egy aritmetikai kifejezést kiértékelni. Ugyanezt az eredményt kapod a következővel:

echo $[1+1]

Ha törtekre vagy más matematikai jellegű dolgokra van szükséged, vagy egyszerűen csak ezt szeretnéd használni, akkor bc-vel is kiértékelheted az aritmetikai kifejezéseket.

Ha beírom a parancssorba, hogy "echo $[3/4]", akkor 0-t kapok, mivel a bash csak egészekkel dolgozik. Az "echo 3/4|bc -l" parancsra viszont már helyes eredményt ad, azaz 0.75-öt.

10.3 A bash felkutatása

Mike levele nyomán (nézd meg a köszönetnyilvánítást):

Eddig mindig a "#!/bin/bash" formát használtuk, most itt van pár példa arra, hogyan tudhatod meg a bash konkrét elérési útját.

A "locate bash" az egyik legjobb módszer erre a feladatra, de nem minden rendszeren van locate.

A gyökérből kiadott "find ./ -name bash" legtöbbször működik.

A lehetséges helyek, ahol nézelődhetsz:

ls -l /bin/bash

ls -l /sbin/bash

ls -l /usr/local/bin/bash

ls -l /usr/bin/bash

ls -l /usr/sbin/bash

ls -l /usr/local/sbin/bash

(Most kapásból nem jut több eszembe... A legtöbb rendszeren

ezen könyvtárak valamelyikében lesz).

Vagy megpróbálhatod a "which bash" parancsot is.

10.4 Program visszatérési értékének kiderítése

Bash-ban a programok visszatérési értéke egy speciális változóban tárolódik, melynek neve $?.

A következő példa bemutatja, hogyan deríthetjük ki egy program visszatérési értékét. Feltételezem, hogy a *dada* nevű könyvtár nem létezik. (Ez szintén Mike javaslatára került ide.)

```
#!/bin/bash
cd /dada &> /dev/null
echo rv: $?
cd $(pwd) &> /dev/null
echo rv: $?
```

10.5 Parancs kimenetének tárolása

A következő szkript listázza az összes táblát az összes adatbázisból (felteszem, hogy MySQL-t használsz). Megváltoztathatod a "mysql" parancsot úgy, hogy valódi felhasználónevet és jelszót használsz.

```
#!/bin/bash
DBS=‘mysql -uroot  -e"show databases"‘
for b in $DBS ;
do
```

```
        mysql -uroot -e"show tables from $b"
done
```

10.6 Több forrásfájl használata

A "source" parancs segítségével több fájlt is használhatsz egyszerre.

__TO-DO__

11 Táblázatok

11.1 Sztring összehasonlító operátorok

(1) s1 = s2

(2) s1 != s2

(3) s1 < s2

(4) s1 > s2

(5) -n s1

(6) -z s1

(1) s1 megegyezik s2-vel

(2) s1 nem egyezik meg s2-vel

(3) __TO-DO__

(4) __TO-DO__

(5) s1 nem nulla (egy vagy több karakterből áll)

(6) s1 nulla

11.2 Példák sztingek összehasonlítására

Két sztring összehasonlítása.

```
#!/bin/bash
S1='string'
S2='String'
if [ $S1=$S2 ];
then
        echo "S1('$S1') nem egyenlő S2('$S2')-vel"
fi
if [ $S1=$S1 ];
then
        echo "S1('$S1') egyenlő S1('$S1')-vel"
fi
```

Most idézek egy részt Andreas Beck leveléből, az *if [$1 = $2]* alakú utasításokkal kapcsolatban.

Nem túl jó ötlet a fenti példában is látható módszert használni, mert ha a $S1 vagy a $S2 üres, értelmezési hibát kapunk. Helyette használjuk inkább az x$1=x$2 vagy a "$1"="$2" formát.

11.3 Aritmetikai operátorok

+

-

*

/

% (maradék)

11.4 Aritmetikai relációs operátorok

-lt (<)

-gt (>)

-le (<=)

-ge (>=)

-eq (==)

-ne (!=)

A C programozók egyszerűen össze tudják vetni ezen operátorokat a zárójelek közt megadottakkal.

11.5 Hasznos parancsok

Ezt a részt Kees újraírta (nézd meg a köszönetnyilvánítást...).

Néhány ezek közül a programok közül egy komplett programozási nyelvet takar. Ezen parancsok esetén csak az alapokat ismertetjük. Ha részletes leírásra is szükséged van nézd meg a parancsok kézikönyv oldalait (man pages).

sed (folyam szerkesztő)

A sed egy nem interaktív szerkesztőprogram. Ez azt jelenti, hogy a fájlok módosítása nem úgy megy, hogy mozgatod a kurzort a képernyőn. Itt szerkesztő utasításokból álló szkripteket használunk és egy fájlnevet, amit szerkeszteni akarunk. A sed tulajdonképpen egy szűrőnek is felfogható. Nézzük az alábbi példákat:

```
$sed 's/mit_cseréljek/mire_cseréljem/g' /tmp/dummy
```

A sed kicseréli a "mit_cseréljek" szöveget a "mire_cseréljem" szöveggel, és a bemenetét a /tmp/dummy fájlból veszi. A végeredmény az stdout-ra lesz küldve (ami alapesetben a konzol), de ha a sor végéhez írod, hogy "> fájlnév", akkor a sed kimenetét a "fájlnév" nevű fájlba küldi.

```
$sed 12, 18d /tmp/dummy
```

A sed, a 12. és 18. sorok közti rész kivételével kiírja a fájl tartalmát. Az eredeti fájl nem változik.

awk (fájlok módosítása, szöveg kinyerése és szerkesztése)

Az AWK-nak számos implementációja létezik (a legismertebbek a GNU-s gawk és az ún. "új awk", a mawk). Az alapelv egyszerű: az AWK mintát keres és minden illeszkedő mintára végrehajt valamilyen műveletet.

Megint létrehoztam egy dummy nevű fájlt, ami a következő sorokból áll:

"test123

test

tteesstt"

```
$awk '/test/ {print}' /tmp/dummy
```

test123

test

A minta, amit az AWK keres, a "test". Ha a /tmp/dummy fájlban talált egy sort, amiben szerepelt a "test" szó, akkor végrehajtotta rá a "print" műveletet.

```
$awk '/test/ {i=i+1} END {print i}' /tmp/dummy
```

3

Ha sokféle mintára keresel, cseréld ki az idézőjelek közti szöveget "-f file.awk"-ra és a mintákat, végrehajtandó tevékenységeket írd a "file.awk" nevű fájlba.

grep (a keresési mintára illeszkedő sorok kiírása)

Az előző fejezetekben már jó néhány grep parancsot láttunk, amelyek az illeszkedő sorokat írták ki. De a grep többre is képes.

```
$grep "ezt keressük" /var/log/messages -c
```

12

Az "ezt keressük" szöveg 12-szer szerepelt a /var/log/messages nevű fájlban.

[Jó, ez csak egy kitalált példa, a /var/log/messages meg lett buherálva. :-)]

wc (sorok, szavak, bájtok megszámolása)

A következő példában mást kapunk, mint amit várunk. A dummy nevű fájl, amit most használunk, a következő szöveget tartalmazza: *"bash bevezetés HOWTO teszt fájl"*

```
$wc --words --lines --bytes /tmp/dummy
```

2 5 34 /tmp/dummy

A wc nem foglalkozik a paraméterek sorrendjével. Mindig egy előre megadott sorrendben írja ki őket, amint azt te is láthatod: (sorok)(szavak)(bájtok)(fájlnév).

sort (szöveges fájl sorainak rendezése)

Most a dummy fájl a következőket tartalmazza:

"b

c

a"

```
$sort /tmp/dummy
```

A kimenet kábé így néz ki:

a

b

c

A parancsok nem lehetnek ilyen egyszerűek :-)

bc (egy számológépes programozási nyelv)

A bc a parancssorból kapja meg a számítási feladatokat (fájlból, nem pedig átirányítás vagy cső útján), esetleg egy felhasználói felületről. A következőkben példákon keresztül bemutatunk néhány parancsot.

A bc-t a -q paraméterrel indítom, hogy ne írja ki az üdvözlő szöveget.

```
$bc -q
```

1 == 5

0

0.05 == 0.05

1

5 != 5

0

2 ^ 8

256

sqrt(9)

3

while (i != 9) {

i = i + 1;

print i

}

123456789

quit

tput (terminál inicializálására vagy a terminfo adatbázis lekérdezésére)

Egy kis példa a tput képességeinek demonstrálására:

```
$tput cup 10 4
```

A kurzor a (y10,x4)-es pozícióban jelenik meg.

```
$tput reset
```

Törli a képernyőt, és a kurzor az (y1,x1) pozícióba kerül. Az (y0,x0) a bal felső sarok koordinátája.

```
$tput cols
```

80

Az x tengely irányában maximálisan elhelyezhető karakterek száma.

Nagyon ajánlott, hogy legalább ezen programok használatával tisztában legyél. Millió olyan kis program van, amikkel csodákra lehetsz képes a parancssorban.

[Néhány példa kézikönyv oldalból vagy GYIK-ből származik.]

12 Még néhány szkript

12.1 Parancs végrehajtása a könyvtárban lévő összes fájlra

12.2 Példa: egyszerű backup szkript (egy kicsivel jobb verzió)

```
#!/bin/bash
SRCD="/home/"
TGTD="/var/backups/"
OF=home-$(date +%Y%m%d).tgz
tar -cZf $TGTD$OF $SRCD
```

12.3 Fájl átnevező program

```
#!/bin/sh
# renna: egyszerre több fájl átnevezése különböző szabályok szerint
# írta: Felix Hudson (2000. január)

# először is megnézzük, hogy a program milyen "üzemmódban" van
# ha az első paraméter ($1) alapján valamelyik feltétel teljesül végrehajtjuk
# a hozzá tartozó programrészt és kilépünk

# előtag (prefix) szerint történő átnevezés?
if [ $1 = p ]; then

# most megszabadulunk a módot tartalmazó változótól ($1) és a prefixtől ($2)
  prefix=$2 ; shift ; shift

# megnézzük, adtak-e meg fájlnevet
# ha nem, jobb ha nem csinálunk semmit, minthogy nemlétező fájlokat
# nevezzünk át

  if [$1 = ]; then
     echo "nem adtál meg fájlnevet"
     exit 0
  fi

# ez a for ciklus végigmegy a megadott fájlokon és mindet
# egyenként átnevezi
  for file in $*
```

```
    do
    mv ${file} $prefix$file
  done

# kilépünk a programból
  exit 0
fi

# utótag (suffix) szerinti átnevezés?
# az ide tartozó rész tulajdonképpen megegyezik az előzővel
# nézd meg az ottani megjegyzéseket
if [ $1 = s ]; then
  suffix=$2 ; shift ; shift

   if [$1 = ]; then
    echo "nem adtál meg fájlnevet"
   exit 0
   fi

 for file in $*
  do
   mv ${file} $file$suffix
 done

 exit 0
fi

# helyettesítéses átnevezés?
if [ $1 = r ]; then

  shift

# ezt a részt azért raktam bele, hogy ne tegyük tönkre a felhasználó egyik
# állományát se, ha helytelenek a paraméterek
# ez csak egy biztonsági intézkedés

  if [ $# -lt 3 ] ; then
    echo "használat: renna r [kifejezés] [helyettesítés] fájlok... "
    exit 0
  fi

# továbblépünk a paraméterlistán
  OLD=$1 ; NEW=$2 ; shift ; shift

# Ez a for ciklus végigmegy az összes fájlon amit a programnak átadtak,
# és egyenként átnevezi őket a "sed" program segítségével.
# A "sed" egyszerű parancssori program, ami értelmezi a bemenetet és
# kicseréli a beállított kifejezést egy adott szövegre.
# Mi itt a fájl nevét adjuk át neki (standard bemenetként), és kicseréljük
# benne a kifejezésnek megfelelő szövegrészeket.

  for file in $*
  do
    new=‘echo ${file} | sed s/${OLD}/${NEW}/g‘
    mv ${file} $new
```

```
  done
exit 0
fi

# Ha idáig eljutottunk, akkor semmi használhatót nem adtak át a programnak
# ezért kiírjuk a felhasználónak, hogyan használja ezt a szkriptet.
echo "használat:"
echo " renna p [előtag] fájlok.."
echo " renna s [utótag] fájlok.."
echo " renna r [kifejezés] [helyettesítés] fájlok.."
exit 0

#ennyi!
```

12.4 Átnevező program (egyszerű változat)

```
#!/bin/bash
# renames.sh
# egyszerű átnevező program

criteria=$1
re_match=$2
replace=$3

for i in $( ls *$criteria* );
do
    src=$i
    tgt=$(echo $i | sed -e "s/$re_match/$replace/")
    mv $src $tgt
done
```

13 Ha gond van... (hibakeresés)

13.1 A bash meghívásának módjai

Jó dolog, ha az első sorban a következő szerepel:

```
#!/bin/bash -x
```

Ennek hatására érdekes kimeneti információkat kaphatunk.

14 A dokumentumról

Szabadon küldhetsz javaslatokat/hibajavításokat vagy bármi olyat, amiről úgy gondolod, hogy érdekes és szívesen látnád a dokumentumban. Megpróbálom majd olyan gyakran frissíteni a leírást, amilyen gyorsan csak tudom.

14.1 (nincs) garancia

A dokumentummal kapcsolatban semmiféle garancia nincs.

14.2 Fordítások

Olasz: by William Ghelfi (wizzy at tiscalinet.it)

itt a cím <http://web.tiscalinet.it/penguin_rules>

Francia: by Laurent Martelli

nincs cím <http://>

Koreai: Minseok Park

http://kldp.org <http://kldp.org>

Koreai: Chun Hye Jin

ismeretlen <>

Spanyol: ismeretlen

http://www.insflug.org <http://www.insflug.org>

Lehet, hogy több fordítás is van, de én nem tudok róla. Ha neked ilyen van, kérlek küldd el, hogy frissíthessem ezt a részt.

14.3 Köszönet

- Azoknak, akik más nyelvekre fordították a dokumentumot (lásd: előző rész).
- Nathan Hurstnek a sok hibajavításért.
- Jon Abbottnak az aritmetikai kifejezések kiértékeléséhez fűzött megjegyzéseiért.
- Felix Hudsonnak a *renna* szkriptért
- Kees van den Broeknek (számos hibajavításért és a "Hasznos parancsok" című fejezet újraírásáért)
- Mikenak (pink), a bash felkutatásával és fájlok tesztelésével kapcsolatos javaslataiért.
- Fieshnek, a ciklusokkal foglalkozó részben tett hasznos megjegyzéséért.
- Lionnak, akinek javaslatára megemlítettünk egy gyakori hibát (./hello.sh: Command not found).
- Andreas Becknek a hibajavításokért és megjegyzésekért.

14.4 Történet

Új fordítások és lényegi javítások.

A Kess által átírt hasznos parancsok rész hozzáadása.

Hibajavítások, javaslatok.

Példák a sztring összehasonlításhoz.

v0.8 Elhagyom a verziószámozást, rájöttem, hogy a dátum is elég.

v0.7 Számos hibajavítás és pár régi TO-DO rész megírása.

v0.6 Kisebb javítások.

v0.5 Az átirányítással foglalkozó rész hozzáadása.

v0.4 A volt főnökömnek köszönhetően a doksi eltűnt a régi helyéről, most a www.linuxdoc.org címen érhető el.

előtte: Nem emlékszem, nem használtam se rcs-t, se cvs-t. :(

14.5 További források

Bevezetés a bash programozásba (BE alatt)

http://org.laol.net/lamug/beforever/bashtut.htm <http://org.laol.net/lamug/beforever/bashtut.htm>

Bourne shell programozás

http://207.213.123.70/book/ <http://207.213.123.70/book/>

14.6 További források magyarul

(Ezeket a forrásokat a lektor gyűjtötte össze.)

Büki András: "UNIX/Linux héjprogramozás" (ISBN: 963 9301 10 8)

Bourne shell programozás

http://www.inf.u-szeged.hu/~bmse/unix/unix3.html <http://www.inf.u-szeged.hu/~bmse/unix/unix3.html>

Shell programozás

http://shell-script-programing.sync.hu/SHELL_PROGRAMOZAS.htm <http://shell-script-programing.sync.hu/SHELL_PROGRAMOZAS.htm>

Shell programozás

http://www.fsz.bme.hu/~szebi/slides/U3/sld010.htm <http://www.fsz.bme.hu/~szebi/slides/U3/sld010.htm>

Shell programozás

http://www.szabilinux.hu/ufi/10.html <http://www.szabilinux.hu/ufi/10.html>