I. Alapok

Interaktív shell-ben vagy shell-scriptben megadott karaktersorozat feldolgozásakor az első lépés a szavakra tördelés. A szavakra tördelés a következő metakarakterek mentén zajlik: **| & ; ( ) < > ¬ space** és **tabulátor.**

- **Megjegyzések**

echo Hello # World > Hello

- **Stat parancs**

stat ./script.sh << Fontos információkat ad meg a fájlról, pl.: **magic number** >>

- **File parancs**

file ./script.sh > Bourne-Again shell script text executable << tartalmat néz >>

II. Egyszerű parancsok

IFS=: **read** elso masodik maradek
[1] [2] [3]

echo A B C > fajl.txt
[2] [3] [4]

[1] Változó hozzárendelés
[2] Parancs
[3] Argumentum
[4] Átirányítás

Az egyszerű parancs elemeit space, illetve tabulátor karakterek választják el egymástól. Végét olyan metakarakter jelzi, mely az egyszerű parancsban nem szerepelhet: **| & ; ( ), ¬.** Az egyszerű parancs visszatérési értéke megegyezik a parancs kilépési kódjával. A 0 kilépési kód a sikeres futást jelzi, míg az ettől különböző értékek a sikertelen futás okát (hibakódját) adják meg.

III. Csővezetékek

A csővezeték egyszerű vagy összetett parancs vagy parancsok összefűzésével készíthető. (Így a bash egyetlen egyszerű parancsot is csővezetékként definiál.) A csővezeték elemeit a | vagy a |& token választja el egymástól.

A | token-nel elválasztott parancsok végrehajtásakor a bal oldali parancs kimenete átirányításra kerül a jobb oldali parancs bemenetére.

A **|&** token-nel elválasztott parancsok végrehajtásakor a bal oldali parancs kimenete a hibakimenettel együtt átirányításra kerül a jobb oldali parancs bemenetére.

A csővezeték visszatérési értéke alapesetben megegyezik az utoljára végrehajtott parancs visszatérési értékével.

time egyszerű parancs | egyszerű parancs | egyszerű parancs | …

echo Hello**; echo** $? > **Hello**
> **0**

cp; echo $? > **Hibaüzenet**
> **1**

A $? Változó megadja az utolsó csővezeték visszatérési értékét.

IV. Listák

A lista csővezetékek egymástól az alábbi token-ekkel elválasztott sorozata: **& || ; && ¬**.

A **;** és ¬ karakterekkel tagolt csővezetékek végrehajtása szekvenciálisan, balról jobbra történik Visszatérési értéke egybeesik az utoljára futtatott csővezeték visszatérési értékével. A **&** karakterrel zárt csővezetékek futtatása a háttérben történik, visszatérési értéke 0 (hiszen befejezését nem várja meg a shell a következő pipeline futtatása előtt). A lista **hibaüzenetektől függetlenül** végrehajtódik.

A **&&** és || karakterek rövidzár operátorok, nem minden esetben hajtódik végre a jobb oldali csővezeték.

Egy lista opcionálisan **&**, **;** vagy **¬** karakterrel záródhat.

V. Összetett utasítások

Zárójelezés segítségével az alábbi formában készíthetünk összetett utasítást:

(lista) vagy **{ lista; }** vagy **{ lista ¬ }**

A kerek zárójelpár használata esetén a lista végrehajtása subshell-ben történik, változói lokálisak maradnak.

A kapcsos zárójelpár használatakor a lista végrehajtása az aktuális shellben zajlik.

A lezáró **;** illetve **¬** karakterekre azért van szükség, mert a kapcsos zárójel nem metakarakter. Ezért nem írhatók a zárójelek a listával egybe, elhatárolatlanul.

VI. Paraméterek

- Parancssori argumentumok

Azonosítójuk pozitív egész számok, de vannak speciális nevek is.

echo $0	Parancs neve.
echo $1	Első paraméter értéke.
echo ${10}	Tizedik paraméter értéke
echo $#	Paraméterek száma.
echo $*	Az összes paraméter felsorolása szóközzel elválasztva.
echo $?	0, ha a script helyesen futott le az utolsó argumentumnál.

A parancs paramétereinek felsorolásakor a szóköz jelenti a paraméterek elhatárolását, ezért ha például egy ember teljes nevét akarjuk paraméternek adni, a szóköz elhatároló szerepét meg kell szűntetnünk a \ jellel. Pl.: Hack\ Elek. Hosszabb szövegek paraméterbeli átadásához hatékonyabb, ha a szöveget **'** vagy **"** karakterek közé írjuk.

A **'** karakterek minden speciális karakter jelentését elveszik és a szöveget úgy írják le, mint ahogy olvassuk.
A **"** karakterek azonban meghagyják a **$** és a ` jelek speciális jelentését, így a változók továbbra is elérhetőek.

- Változók

Betűvel (53-féle a _ miatt) kezdődik és betűvel vagy számmal folytatódhat.

Ha egy változót deklarálunk, az lokálisan látható lesz, de futtatandó script-eken belül már nem.

ir.sh

```
#! /bin/bash
echo $1
```

```
alma=finom
./ir.sh alma
```

> finom

ir.sh

```
#! /bin/bash
echo $alma
```

```
alma=finom
./ir.sh
```

<< üres sor >>

Ezt a problémát úgy küszöbölhetjük ki, hogy exportáljuk a változót, hogy az mindenütt látható maradjon.

ir.sh

```
#! /bin/bash
echo $alma
```

```
export alma=szuper
./ir.sh
```

> szuper

Változót deklarálhatunk úgy is, hogy megadjuk a tulajdonságait.

declare –i szam=5 << szam egy pozitív egész típusú változó, melynek értéke 5 >>

- l: kisbetűs sztring
- u: nagybetűs sztring
- x: exportált
- r: csak olvasható

declare +u szam << a + elveszi, a - hozzádja a változóhoz a tulajdonságot >>

Ahhoz, hogy egy változót megszűntessünk az **unset** parancsot kell használnunk.

VII. Aritmetikai kifejezések

Összetett utasítás formájában bash-ben aritmetikai kifejezés kiértékelése az alábbi formában adható meg:

((aritmetikai-kifejezés))

Az utasítás visszatérési értéke pontosan akkor 0, ha az aritmetikai kifejezést kiértékelve 0-tól különböző értéket kapunk. Az aritmetikai kifejezésben használt operátorok és precedenciájuk lényegében megegyezik a C nyelvben használatosakkal. Itt azonban minden kifejezés típusa rögzített méretű egész, továbbá operandusként shell változókat is használhatunk.

VIII. Logikai kifejezések

Összetett utasítás formájában bash-ben logikai kifejezés kiértékelése az alábbi formában adható meg:

[[logikai-kifejezés]]

Az utasítás visszatérési értéke 1, ha a logikai kifejezést kiértékelve 0 értéket kapunk. Fontos megjegyezni, hogy a 0 logikai igazat, az 1 logikai hamist jelent.

[[13 -gt 5]]; **echo** $?	> 0 (igaz)
[[13 -lt 5]]; **echo** $?	> 1 (hamis)
if [[ab > a]]; **then echo** Nagyobb; **else echo** Kisebb; **fi**	> Nagyobb (a kif. értéke 0 [logikai igaz])
if ((13 – 13)); **then echo** Nemulla; **else echo** Nulla; **fi**	> Nulla (az ar. kif. értéke 0, amely v.é.: 1])

IX. Állományok vizsgálata

Állományokkal kapcsolatosan számos logikai vizsgálatra van lehetőség. A teljesség igénye nélkül, a leggyakrabban használt logikai kifejezések a következők:

[[-e fájl]]	A fájl létezik.
[[-d fájl]]	A fájl egy létező katalógus.
[[-f fájl]]	A fájl közönséges állomány.
[[-c fájl]]	A fájl karakteres állomány.
[[-s fájl]]	A fájl létezik és nem üres.
[[-r / -w / -x fájl]]	A fájl létezik és olvasható / írható / futtatható.
[[fájl1 -nt fájl2]]	Fájl2 nem létezik vagy régebbi mint fájl1.
[[fájl1 -ot fájl2]]	Fájl2 nem létezik vagy újabb mint fájl1.
[[fájl1 -ef fájl2]]	Fájl1 és fájl2 egybeesik.

X. Karaktersorozatok összehasonlítása

[[-z $string]]	A string karakterlánc hossza 0.
[[-n $string]]	A string karakterlánc hossza nem 0.
[[$string1 == $string2]]	A karakterláncok megegyeznek.
[[$string1 != $string2]]	A karakterláncok különböznek.
[[$string1 < $string2]]	String1 lexikografikusan megelőzi string2-t.
[[$string1 > $string2]]	String2 lexikografikusan megelőzi string1-et.

XI. Egészek összehasonlítása

Amennyiben **op1** és **op2** egész számokat jelöl, úgy a következő összehasonlítások tehetők meg logikai kifejezés formájában:

[[$op1 -eq $op2]]	Op1 és op2 egyenlőek.
[[$op1 -ne $op2]]	Op1 és op2 különböznek.
[[$op1 -lt $op2]]	Op1 kisebb, mint op2.
[[$op1 -le $op2]]	Op1 kisebb vagy egyenlő, mint op2.
[[$op1 -gt $op2]]	Op1 nagyobb, mint op2.
[[$op1 -ge $op2]]	Op1 nagyobb vagy egyenlő, mint op2.

XII. Vezérlési szerkezetek

for name; **do** lista; **done**

count.sh

```
declare –i count=0
for i
do
        (( count+=i ))
done
echo $count
```

```
./count.sh 1 2 3
> 6
```

---

for name **in** word; **do** lista; **done**

list.sh

```
for name in alma *
do
        echo [$name]
done
```

```
./list.sh
> [alma]
> [fájlnév]
> …
```

---

for ((exp1; exp2; exp3)); **do** lista; **done**

negyzet.sh

```
for (( i=1; i<=10; i++ ))
do
        echo $i*$i $'\t'= $(( i * i ))
done
```

```
./negyzet.sh
> 1*1   = 1
> 2*2   = 4
> …
```

---

while lista; **do** lista; **done**

countdown.sh

```
declare –i i=$1
while (( i > 0 )); do
        echo $i
        ((i--))
done
```

```
./coutdown.sh 4
> 4
> 3
> 2
> 1
```

---

until lista; **do** lista; **done**

countdown.sh

```
declare –i i=$1
until (( i == 0 )) do
        echo $i
        ((i--))
done
```

```
./coutdown.sh 4
> 4
> 3
> 2
> 1
```

---

if lista; **then** lista; **else** lista; **fi**

equals.sh

```
if [[ $1 -eq $2 ]]
  then echo Egyenlő
  else echo Nem egyenlő
fi
```

```
./equals.sh 5 5
> Egyenlő
```

XII + I. Érdekességek

- Felsorolás

```
echo a{b,c,d}e                    > abe ace ade
echo {1..3}{a..c}                 > 1a 1b 1c 2a 2b 2c 3a 3b 3c
```

- Változók

```
alma=finom; echo ${#alma}         > 5
alma=korte; echo ${alma:3:2}      > te
echo ${alma:=érték}               > érték (ha alma még nem volt deklarálva)
echo ${alma//a/e}                 > elme (az a betű minden előfordulását lecseréli e betűre)
echo ${alma^^}                    > ALMA (ha alma értéke alma)
echo ${alma,,}                    > alma (ha alma értéke ALMA)

echo A${alma}B                    > AxB (ha alma értéke x)
echo A$alma\B                     > AxB (ha alma értéke x)
echo A$almaB                      > A
echo A\$alma\B                    > A$almaB
```

- Case

```
case $1 in                        > Ha a paraméter szám, akkor kilép.
    "" | *[!0-9.]* | *[!0-9] ) exit 1 ;;
esac
```

- Read

```
echo A; echo B | echo Erdemény: `cat`     > A
                                          > Eredmény: B

{ echo A; echo B; } | echo Eredmény: `cat`  > Eredmény: A B (bár cat sorokat ír, az echo csak sorba ír)
```

- Set

```
var="10 11 12"                    > 3
set -- $var
echo $#
```

- Ekvivalens kifejezések

```
echo 'ls';                        echo $(ls);
```

- Függvények

```
function miez                     ./shellscript.sh Lovacska Kecske
{
    echo Ez egy $1                > Ez egy Kecske
}

miez $2 $1
```

- Echo és Cat

```
{ echo alma; echo korte; } | cat          > alma\nkörte
{ echo alma; echo korte; } | echo `cat`   > alma körte
```