

Héjprogramozás Linux alatt – a sed használata (6. rész)

Az előző részben az awk nyelvvel ismerkedtünk meg, amellyel számos különböző szövegátalakítási és matematikai műveletet végezhetünk el a héjprogramokban. Ebben a részben egy másik szövegkezelő eszközzel, a sed segédprogramról lesz szó.

A sed (stream editor) tulajdonképpen egy programozható szövegszerkesztő, ami a szabványos bemenetére érkező szöveget „röptében” képes átalakítani (természetesen fájlból is tud olvasni).

Működésének lényege, hogy a feldolgozandó szöveget soronként egy átmeneti tárbba, az úgynevezett mintatérbe olvassa be, szabályos kifejezések alapján megkeres benne bizonyos részeket, majd elvégzi rajtuk az egybetűs parancsok formájában megadott műveleteket.

Bár a sed így, a XXI. század hajnalán sokak számára igencsak „fapadosnak” tűnhet, ha megtanulunk bánni vele, csodákra képes. Kezdők számára a legriasztóbb vonása a szabályos kifejezések használata, ami kétségkívül a Unix egyik legnehezebben megtanulható részét képezi. Ezzel kapcsolatban biztatásul is csak annyit mondhatok, hogy feltétlenül érdemes elsajátítani. A szabályos kifejezések teljes ismertetésére ez a rövid cikk messze nem elég, ugyanakkor tömör és világos leírást találhatunk róluk gyakorlatilag valamennyi, a Unix operációs rendszerről szóló tankönyvben.

A sed-et leggyakrabban egy másik programtól érkező kimenet feldolgozására használjuk, valahogy így:

```
... | sed 'program' | ...
```

Az egyszeres idézőjelek között megadott 'program' itt a sed saját nyelvén írt szövegfeldolgozási utasítássorozatot jelenti, ami egybetűs parancsokat és szabályos kifejezéseket tartalmaz. Ez a program több sorban több feldolgozási lépést is tartalmazhat, amelyek a megadás sorrendjében mennek végbe. (A későbbi utasítások már a korábbi átalakítások eredményét kezelik, és nem az eredeti szöveget.) A sed programok egy sora legalább a legáltalánosabb formájában a következőképpen néz ki:

```
<cím1>,<cím2> parancs
```

Itt <cím1> és <cím2> egy-egy szám vagy szabályos kifejezés lehet. Ha számot adunk meg, az a bemenet adott sorszámú sorának feldolgozását jelenti, ha szabályos kifejezést (ezt két / – perjel – közé kell zárni), akkor a program minden olyan sorra léfut, amelyre a kifejezés illeszkedik. Ennek megfelelően a

```
25 egybetűs_parancs
```

utasítás csak a bemenet 25. sorát fogja érinteni, míg a

```
/[0-9]/ egybetűs_parancs
```

programsor minden olyan szövegsort érinteni fog, amelyben legalább egy számjegy van.

A sed működése közben amolyan átfolyóként viselkedik, ami azt jelenti, hogy azokat a sorokat is kiküldi a kimenetére,

amelyekkel semmi dolga nem akadt. A használható parancsokat *táblázatban* foglaltam össze. Ezek működéséről, illetve a szabályos kifejezések használatáról a megfelelő sűgőoldalakon találhatunk bővebb ismertetést. (Mint korábban is említettem, kezdők számára erősen ajánlott valamilyen Unix-tankönyv böngészése is.)

Tizedes törtek

Ennyi bevezető után nézzünk egy egyszerű példát! Mint azt nyilván az olvasó is tudja, a

magyar szövegekben előforduló tizedes törtekben a nyugaton szokásos tizedespont helyett tizedesvesszőt használunk. Ugyanakkor gyakori, hogy egy számsorozat vagy egy táblázat egy program kimeneteként keletkezik, s így mégis pontokat tartalmaz. Írjunk most egy olyan héjprogramot, amelyik „magyarosítja” a tizedes törtek írásmódját! Bár a feladat első látásra egészen egyszerűnek tűnhet, ne feledkezzünk meg róla, hogy egy szövegben sokféle helyen szerepelhet pont (például e mondat végén is). Az tehát biztosan nem jó megoldás, ha az összeset egyszerűen vesszőre cseréljük le.

Az első feladat a tizedespont egyfajta logikai meghatározása, aminek alapján megírhatjuk a behatároláshoz szükséges szabályos kifejezést. Nos, miről is ismerszik meg egy jó családból való tizedespont? Két számjegy határolja, vagyis illeszkedik rá a

```
[0-9]\.[0-9]
```

szabályos kifejezés. Ezzel a lényeg tulajdonképpen meg is van, de akad itt még néhány nehézség. Csábító ugyan a gondolat, de a fenti mintát nem használhatjuk egy cserélő utasításban a következő módon:

```
sed 's/[0-9]\.[0-9]/,/g'
```

Az a baj ezzel a megoldással, hogy a csere a teljes illeszkedő részre vonatkozik, vagyis a tizedespontot megelőző és az azt követő számjegyet is törölni fogja (a sor végén szereplő „g” egy jelentésmódosító kapcsoló, ami az összes lehetséges illeszkedés cseréjét írja elő). Az sem teljesen jó megoldás, ha az előbbi szabályos kifejezést a feldolgozandó sor címzésére használjuk:

```
sed '/[0-9]\.[0-9]/ s/\./,/g'
```

A sed egybetűs parancsai

p	Kiíratás
d	Törlés
s	Helyettesítés
a	Hozzáfűzés
i	Beszúrás
c	A mintatér cseréje
y	Karakterek cseréje

Ilyenkor a helyettesítés a tizedes törtet is tartalmazó sorokban minden pontot érinteni fog. A helyes programnak a következőket kell tartalmaznia:

1. Megkeressük a két számjegy által határolt pontokat.
2. A pont mindkét oldalán megkeressük és megjegyezzük a leghosszabb, csak számjegyekből álló karakterláncot.
3. A két karakterláncot önmagával, a köztük levő pontot pedig vesszővel helyettesítjük.

Ennek a műveletsornak a következő `sed` program felel meg:

```
sed 's/\([0-9][0-9]*\)\.\([0-9][0-9]*\)/\1\,\2/g'
```

Itt bizony már látszik a szabályos kifejezéseknek a kezdő felhasználókat gyakran megbabonázó szépsége. Lássuk, miről is van szó. Valahol a minta közepénél felismerhető a keresett tizedespont (`\.`). Ennek mindkét oldalán kerek zárójelek között szerepel a `[0-9][0-9]*` szabályos kifejezés, ami emberi nyelvre lefordítva egyszerűen azt jelenti, hogy 'legalább egy számjegy'. (A szabályos kifejezésekben a `*` jelentésmódosító műveleti jel (operátor) „nullasoros” illeszkedést is megenged, ezért kellett kétszer kiírni a `[0-9]` tartományt.) A kerek zárójelek hatására a `sed` nemcsak megkeresi a pont két oldalán található, csak számjegyekből álló karakterláncot (a tizedes tört egészrészét és törtrészét), hanem meg is jegyzi azokat két átmeneti tárban. Ezekre hivatkozunk a helyettesítésnél a `\1` és `\2` szimbólumokkal.

Kész programunk némi „szokásos körítéssel” kiegészítve a következőképpen fest:

```
1: #!/bin/sh
2: PROGRAMNEV=`basename $0`
3: if [ $# -ne 1 ]
4: then
5:   echo "Használat: $PROGRAMNEV fájlnev"
6:   exit 1
7: fi
8: if [ -f $1 ]
9: then
10:  cat $1 | sed
11:  's/\([0-9][0-9]*\)\.\([0-9][0-9]*\)/\1\,\2/g'
12: else
13:  echo "A fájl nem létezik!"
14:  exit 2
15: fi
```

Szűrőként működő héjprogramok

A cikk elején említettem, hogy a `sed` egyik legfontosabb tulajdonsága „átfolyóként” való működése, vagyis hogy a beérkező szöveget akkor is átérteszti, ha semmilyen műveletet nem kellett rajta elvégeznie. Az előbbi héjprogramunk azonban sajnos úgy lett megszerkesztve, hogy általa a `sed`-nek ezt az értékes tulajdonságát elveszítjük. Amennyiben ilyen szűrőként működő tizedespont-kezelésre van szükségünk, bármikor megtehetjük, hogy a megfelelő helyen csak magát az előbb kidolgozott `sed` programot használjuk. Ez azonban érezhetően körülményes, és egyébként is, ki szokott kapásból emlékezni ilyen körmönfont megoldásokra?

Ez az a pont, ahol nagy hasznát vehetjük a `read` parancsnak, ami a héjprogram szabványos bemenetéről egysornyi adatot olvas be az utána megadott héjváltozóba. Ha az olvasás sikeres

volt, visszatérési értéke igaz, ha nem, akkor hamis. Ezt az utóbbi tulajdonságát kihasználva egy ciklusban önműködően érzékelhetjük a bemenet végét, valahogy így:

```
1: #!/bin/sh
2: while read sor
3: do
4:   echo $sor | sed \
5:   's/\([0-9][0-9]*\)\.\([0-9][0-9]*\)/\1\,\2/g'
6: done
```

Ez a program pontosan ugyanúgy működik, mint az előbbi, de a feldolgozandó szöveget nem fájlból veszi, hanem a szabványos bemenetéről. Ha tehát mondjuk *átfolyó.sh*-nak hívják, akkor a következőképpen kell használni:

```
cat feldolgozandó_szoveg.txt | átfolyó.sh
```

A nagy egyesítés

Számos olyan Unix-parancs van, amelyik a szabványos bemenetéről és egy parancssori kapcsolóként megadott fájlból egyaránt képes dolgozni. Ráadásul önműködően felismeri a pillanatnyi helyzetet, vagyis nem kell külön elmagyarázni neki, hogy mit akarunk. Esetünkben is sokkal célszerűbb volna ezt a viselkedésmódot megvalósítani, mint két, valójában azonos feladatot ellátó programot írni. Íme az egyik lehetséges megoldás:

```
1: #!/bin/sh
2: # Szabványos bemenet feldolgozása
3: if [ $# -eq 0 ]
4: then
5:   while read sor
6:   do
7:     echo $sor | sed 's/\([0-9][0-9]*\)\.\([0-9][0-9]*\)/\1\,\2/g'
8:   done
9: # Parancssori fájlok feldolgozása
10: else
11:  for nev
12:  do
13:    cat $nev | sed 's/\([0-9][0-9]*\)\.\([0-9][0-9]*\)/\1\,\2/g'
14:  done
15: fi
```

Ez a program mindenképpen működik, hiszen vagy a szabványos bemenetet (5–8. sorok), vagy a parancssorban megadott valamennyi fájl (11–14. sor) dolgozza fel. (Bár a `for` ciklus feje kissé befejezetlennek tűnhet, valójában nem az. Ha nem adunk meg listát, a ciklus alapértelmezésként önműködően a parancssori kapcsolókon megy végig.) Ha egyáltalán semmit nem adunk meg neki, amivel működhetne, akkor tőlünk várja, hogy gépeljünk neki valamit, akár a többi, hasonló célú Unix-segédprogram.



Büki András (buki.andras@insilico.hu)

Körülbelül kilenc éve dolgozik Linux operációs rendszerrel. Állandó szerzőtársa Prof. Dr. H. V. Kuksinak, akivel a Duna vagy a Tisza partján szoktak az élet és a tudomány viselt dolgairól tőprengeni.