Héjprogramozás Linux alatt (10. rész)

Héjprogramozásról szóló cikksorozatunk utolsó részében a kifinomultabb programozási fogásokból és a különleges megoldásokból mutatunk be néhányat.

Számos olyan Unix-parancs, illetve segédprogram létezik, amelyik képes a rekurzióra, vagyis arra, hogy egy bizonyos műveletet ne csak egy adott könyvtár fájljain, hanem egy teljes könyvtárszerkezeten hajtson végre. A legegyszerűbb példa erre talán a mindenki által ismert „statáriális” törlés, az `rm -rf *` parancs, amely a pillanatnyi könyvtártól kezdődően mindent eltüntet. Némi képzelőerővel egy héjprogramot is felruházhatunk efféle „teljes hatókörű áttekintőképességgel”. Tegyük fel, hogy egy nagyobb programot fejlesztünk, amelynek a kódja több tíz könyvtárban szétszórt (pontosabban rendszerezett…) több száz forrásfájlból áll. Ebben a rengetegben megtalálni valamit – például egy függvény meghatározását – néha egyáltalán nem egyszerű feladat. Nagy hasznunkra lehet a `grep` parancs.
Milyen jó is lenne, ha volna egy „rekurzív” `grep` parancsunk, ami erre is képes lenne (igaz, ami igaz, van ilyen program, a `grep -r` illetve az `rgrep` parancsok pontosan ezt teszik, de a feladat szépsége miatt ezt mégsem hagyhatjuk ki). Nosza, írjuk meg! A legtöbben most valószínűleg többoldalas forráslistára számítanak, pedig a megoldás az összes szokásos körítéssel együtt mindössze ennyi:

```
 1: #!/bin/sh
 2: # Önhívó grep parancs
 3:
 4: PROGRAMNEV=`basename $0`
 5:
 6: if [ $# -ne 1 ]
 7: then
 8:      echo "Használat: $PROGRAMNEV
          ➥<keresendo minta>"
 9:      exit 1
10: fi
11:
12: find . -type f -exec grep "$1" {} \;
```

Tizenkét sor, amiből az első tizeneggyel kapcsolatban valószínűleg már semmit sem kell elmagyaráznunk. A tizenkettedik sor viszont határozottan érdekes. Itt a kereséshez a valamennyi Unix-rendszeren megtalálható `find` segédprogramot használjuk. Ez nemcsak a feltételnek megadott fájlok kikeresésére képes, hanem arra is, hogy egy adott műveletet végrehajtson rajtuk. Ez utóbbit a `-exec` kapcsoló után adhatjuk meg. Az egyszerűség kedvéért nézzük végig egyenként a 12. sor elemeit! A `find` utáni . (pont) azt jelenti, hogy a keresést a pillanatnyi könyvtárban kezdjük. A `-type f` segítségével határoztuk meg, hogy fájlokat keresünk, vagyis a könyvtárak „kimaradnak a játékból”. Az `exec` után látható `grep` pontosan az, aminek látszik: a jól ismert `grep` parancs, amit most a `find` minden olyan fájlra végre fog hajtani, ami a keresési feltételeknek megfelel. A találati listát a {} (az egyszeres kapcsos zárójelkettős) jelöli, vagyis a `find` ezt fogja helyettesíteni a megtalált fájlok listájával. A végrehajtandó parancssor végét a pontosvessző jelzi, amit azért kellett levédenünk, mert a héj szintén értelmezi.

Néhány célszerű kiegészítés

Ez a program valóban úgy működik, mint a klasszikus `grep` parancs, vagyis a teljes könyvtárszerkezeten végighaladva minden fájlból megjeleníti azokat a sorokat, amelyekben szerepel a keresett szó vagy kifejezés. Példánknál maradva, ha csak az a célunk, hogy a forrásfájlokból kikeressük egy változó alapértelmezett értékét vagy megállapítsuk egy függvény értékeinek helyes sorrendjét, akkor ez a megoldás tökéletes. Hogyha azonban arra vagyunk kíváncsiak, hogy hol volt találat, akkor ezzel a kimenettel semmire sem megyünk. Helyette nyilván arra volna szükségünk, hogy a találati sorok helyett a megfelelő fájl neve jelenjen meg. Szerencsére megint nem kell messzire mennünk a megoldásért. Amennyiben átolvassuk a `grep` parancs súgóoldalát, láthatjuk, hogy – számos egyéb mellett – létezik egy `-l` kapcsolója is, amely a szokványos működés helyett éppen azt teszi, amire szükségünk van: fájlnevet jelenít meg a tényleges találat helyett (ez természetesen igaz a `rgrep` és `grep -r` parancsokra is). Nem kell tehát mást tennünk, mint kiegészíteni a 12. sort ezzel a kapcsolóval:

```
find . -type f -exec grep -l "$1" {} \;
```

Megeshet azután, hogy a keresést csak adott típusú fájlokra, például egyedül a fejlécállományokra vagy csak a forrásszövegekre akarjuk korlátozni. Ilyenkor egy második parancssori kapcsolóban (`$2`) megadhatjuk a fájltípusúra jellemző kiterjesztést, és nem típus, hanem név alapján végezhetjük a keresést, valahogy így:

```
find . -name '*.'$2 -exec grep -l "$1" {} \;
```

Itt a parancssorban valóban csak a keresett fájlok kiterjesztését szükséges megadnunk, a "*." előtagot már maga a program illeszti hozzá az utolsó sorban. Természetesen ha nem „belső használatra" terveztük a programot, akkor ez sokak számára zavaró lesz, hiszen a gyanútlan felhasználók önkéntelenül ki fogják írni a teljes alakot. Ez a forma azonban esetünkben már csak azért sem megfelelő, mert ha ezt a csillagot is tartalmazó kapcsolót nem zárják kettős idézőjelek közé, akkor a héj is értelmezi azt, és a csillag helyére be fogja helyettesíteni az adott könyvtár listáját.

Nagy könyvtárszerkezetek kezelése

Az imént bemutatott keresőprogramok két műveletet hajtanak végre: előállítják a megfelelő típusú fájlok neveinek a listáját, majd átadják azt a `grep` parancsnak. Amíg a lista nem teljes, addig a találatokat a programot futtató héj raktározza egy belső átmeneti tárolóban. Bár ez a láthatatlan közbenső művelet normális esetben semmiféle gondot nem jelent, egy általános célú program megvalósításakor mégis jobb, ha nem építünk a felhasználó előrelátására.
És mivel minden átmeneti tár befogadóképessége véges, megtörténhet, hogy a felhasználónk olyan mennyiségű fájlban akar keresést végezni, amelyek listája már túlcsordulást okoz. Ha ez bekövetkezik, akkor rendkívül érdekes helyzet áll elő. A rendszer minden próbálkozásunkra „A paraméterek listája túl hosszú" (***Argument list too long***) üzenettel válaszol, és a megadott műveletet azokon a fájlokon sem hajtja végre, amelyeknek a neveit még képes volt tárolni. Ha például sikerült létrehozunk egy olyan könyvtárat, amelyben a fájlok listájának hossza meghaladta ezt a mágikus határt, akkor még az amúgy mindenható `rm *` parancs sem működik.
Az ilyen helyzetek elkerülésére találták ki az `xargs` parancsot, amellyel úgy hívhatunk meg egy másik programot, hogy a parancssori kapcsolóit (a szükséges kapcsolókat és fájlneveket) az `xargs` szabványos bemenetére küldjük. Az `xargs` ebből és a saját parancssori kapcsolóként megadott másik program nevéből összeállít egy parancssort, és végre is hajtja azt.
A közérthetőség kedvéért lássunk egy egyszerű példát. Az `ls -l *.txt` parancs az `xargs` közbeiktatásával a következőképpen hajtható végre:

```
echo -l *.txt | xargs ls
```

Ennek így semmi értelme sem volna, ha az `xargs` nem nyújtana valami többletet. Ez a bizonyos többletszolgáltatás pedig az, hogy az `xargs` az értéklistát tetszőleges hosszúságú darabokra tudja bontani, és ezzel elkerüli a korábban említett átmeneti tár túlcsordulását.
Az alapértelmezett darabolási hossz eleve úgy van beállítva, hogy idomuljon a kérdéses rendszer igényeihez. Ha tehát semmi más célunk nincs, csak hogy a héjprogramunkat különlegesen nagy bemenetek kezelésre is fel akarjuk készíteni, akkor nem kell egyebet tennünk, mint a feldolgozási sor megfelelő pontjára beilleszteni az `xargs` segédprogramot. Esetünkben ezt a program lelkét jelentő 12. sorban kell megtennünk, a következőképpen:

```
find . -type f print | xargs grep -l "$1"
```

vagy

```
find . -name '*.'$2 print | xargs grep -l "$1"
```

Figyeljük meg, hogy a `grep` futtatását most nem a `find`, hanem az `xargs` végzi. A `find` feladata csupán a fájlok listájának előállítása és kiküldése a szabványos kimenetre.

Közvetett változóhasználat

Ha egy héjváltozó tartalmához akarunk hozzáférni, nem kell egyebet tennünk, mint a neve elé egy $ (dollárjelet) írni, és a rendszer rögtön tudja, mit akarunk. De mi történik akkor, ha a kérdéses változó neve egy másik változóban található, például azért, mert egy feldolgozási műveletsor választotta ki több lehetséges jelölt közül? A helyzet ilyenkor nagyjából olyan, mintha egy magas szintű programnyelvben mutatót használnánk, amellyel közvetett módon férhetünk hozzá a változó tartalmához.
Egy héjprogramban az efféle fogások kivitelezésére több megoldás is kínálkozik.

```
1: #!/bin/sh
2: # Közvetett változóhasználat
3: valtozo=13
4: valtozo_neve="valtozo"
5: parancs="echo $"$valtozo_neve ; eval "$parancs"
6: eval `echo "echo $"$valtozo_neve`
7: eval echo \$$valtozo_neve
8: echo ${!valtozo_neve}
```

Ha a Bash parancsértelmezőt használjuk és lefuttatjuk ezt a programot, négyszer fog megjelenni a 13-as szám. Más héj esetében a negyedik megoldásra (8. sor) hibaüzenetet kapunk, mivel ez a Bash egy különleges szolgáltatását használja.
A 5. sorban a változó nevét tároló változó felhasználásával egy karakterláncban egy parancssort állítunk össze, majd az `eval` segítségével le is futtatjuk.
A 6. sorban tulajdonképpen ugyanezt a megoldást használjuk, de parancsbehelyettesítés segítségével az egész szerkezetet egyetlen sorrá sűrítjük össze.
A 7. sor mutatja talán a legkifinomultabb megoldást: egyszerűen egy literális $ (dollárjelet) illesztünk be a változó nevét tartalmazó változó tartalma elé, és így adjuk át az `eval` parancsnak.
Végül a 8. sor egy különleges, csak a Bash héjnál működő megoldást mutat. A Bash eleve tartalmaz egy programozási szerkezetet a közvetett változóhasználatra, vagyis nála az alapfelszerelés része az, ami az egyszerű héjaknál inkább csak ügyes trükkökkel érhető el.

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 töprengeni.