Kedvenc Bash trükkjeim

Rengeteg gépelést takaríthatunk meg néhány hasznos bash trükkel amelyek a régivágású UNIX héjprogramokból még hiányoztak.

A ***bash***, azaz ***„Bourne again shell”***, a legtöbb ***Linux*** terjesztésben alapértelmezett héjprogram. A ***bash*** héjprogram népszerűsége a ***Linux*** és ***UNIX*** felhasználók között nem a véletlen műve. Igen sok funkciója van amely a felhasználóbarát jelleget és a termelékenységet erősíti. Sajnos nem igazán tudjuk kihasználni ezeket a lehetőségeket, ha a létezésükről sem tudunk.
Amikor először kezdtem el ***Linuxot*** használni, az egyetlen ***bash*** képességet használtam. Nevezetesen, hogy a parancssorban a felfele nyíl segítségével vissza lehet lépkedni a parancstörténetben. Hamarosan további lehetőségekre is rábukkantam, másokat figyelve vagy kérdezősködve. Cikkemben az évek alatt megismert kedvenc ***bash*** trükkjeimet szeretném megosztani mindenkivel.
Ebben a cikkben nem akarjuk a ***bash*** összes képességét összefoglalni; ahhoz egy könyvre lenne szükség, és szép számmal találunk is könyveket, melyek ezzel a témával foglalkoznak. Ilyen például a ***Learning the bash Shell O'Reilly*** könyv. Ebben a cikkben inkább azokat a ***bash*** trükköket szeretném összegyűjteni amelyeket a leggyakrabban használok és amelyek nélkül elveszettnek érezném magam.

Zárójel kiegészítés

Kedvenc ***bash*** trükköm egyértelműen a ***zárójel bővítés (brace expansion)***. A zárójel bővítés vesszővel elválasztott karaktersorozatokból készít nekünk különálló paramétereket. A listát kapcsos-zárójelek, azaz { és } közé tesszük, a vesszők környékén pedig nem hagyunk szóköz karaktereket. Például:

```
$ echo {one,two,red,blue}
one two red blue
```

Az előbbi egyszerű példában bemutatott zárójel bővítés nem igazán nyújt túl sokat a felhasználónak. Tulajdonképpen a fenti példa kettővel több karakter begépelését igényli, mint ha egyszerűen csak ennyit írnánk:

```
echo one two red blue
```

ami azonos eredményt ad. Ugyanakkor a zárójel bővítés rendkívül hasznos tud lenni, ha a zárójelbe foglalt lista közvetlenül egy másik karakter sorozat előtt, mögött vagy annak közepében foglal helyet:

```
$ echo {one,two,red,blue}fish
onefish twofish redfish bluefish
$ echo fish{one,two,red,blue}
fishone fishtwo fishred fishblue
$ echo fi{one,two,red,blue}sh
fionesh fitwosh firedsh fibluesh
```

Figyeljük meg, hogy a zárójelek és a csatlakozó karaktersorozatok között nincsen szóköz karakter. Ha kitesszük a szóközt, a dolgok megbolondulnak:

```
$ echo {one, two, red, blue }fish
{one, two, red, blue }fish
$ echo “{one,two,red,blue} fish”
{one,two,red,blue} fish
```

Ugyanakkor, ha idézőjelet használunk a zárójeleken kívül vagy a listában, akkor szóközöket is beírhatunk:

```
$ echo {“one “,”two “,”red “,”blue “}fish
one fish two fish red fish blue fish
$ echo {one,two,red,blue}” fish”
one fish two fish red fish blue fish
```

A zárójeleket akár egymásba is ágyazhatjuk, de ügyelnünk kell a formára:

```
$ echo {{1,2,3},1,2,3}
1 2 3 1 2 3
$ echo {{1,2,3}1,2,3}
11 21 31 2 3
```

E példák után, biztos sokan azt mondják magukban: ***„Nahát, ezek aztán tényleg ügyes szalontrükkök, de miért jó nekem ez a zárójelbővítés?”*** A zárójel bővítés hasznos lehet, ha biztonsági másolatot akarunk készíteni egy állományról. Ez az egyik kedvenc héjtrükköm. Szinte miden nap használom, amikor egy beállításállományról változtatás előtt másolatot készítek. Például, amikor megváltoztatom az Apache beállításállományt, egy kis gépelést megtakarítva a következőket írhatom:

```
$ cp /etc/httpd/conf/httpd.conf{,.bak}
```

Figyeljük meg hogy semmilyen karakter nem álla a nyitó zárójel és a vessző között. Az ilyesmit teljesen elfogadott és hasznos amikor betűket szeretnénk egy létező fájlnévhez fűz-

ni vagy ha az egyik paraméter részhalmaza a másiknak. Aztán ha később kíváncsi vagyok milyen változtatást végeztem, könnyen megtudhatom a `diff` parancs segítségével a karaktersorozatokat fordított sorrendben megadva a zárójelben:

```
$ diff /etc/httpd/conf/httpd.conf{.bak,}
1050a1051
> # I added this comment earlier
```

Parancs helyettesítés

A másik ***bash*** trükk amit szeretek használni a parancshelyettesítés. A parancshelyettesítés használatához egy szabványos kimenetre író parancsot helyezzünk zárójelek közé, majd a nyitó zárójel elé tegyünk egy dollár jelet, `$(command)`. A parancshelyettesítés nagyon hasznos ha értéket akarunk adni egy változónak. Elég általános a héjprogramokban, ahol gyakori művelet a dátum vagy idő hozzárendelése egy változónévhez. Akkor is hasznos lehet, ha az egyik parancs kimenetét egy másik program paramétereként szeretnénk felhasználni. Például, amikor a dátumot szeretnénk egy változóhoz rendelni, a következőt írjuk:

```
$ date +%d-%b-%Y
12-Mar-2004
$ today=$(date +%d-%b-%Y)
$ echo $today
12-Mar-2004
```

Gyakran használom a parancskiegészítést amikor több ***RPM*** csomagról szeretnék egyszerre információt kapni. Például ha azon ***RPM*** csomagok fájllistájára vagyok kíváncsi amelyek nevében szerepel a `httpd` szó, egyszerűn a következő parancsot adom ki:

```
$ rpm -ql $(rpm -qa | grep httpd)
```

A belső parancs az `rpm -qa | grep httpd`, valamennyi `httpd` szót tartalmazó nevű csomagot listázza. A külső parancs az `rpm -ql`, pedig a csomagban található állományokat jeleníti meg. Most a tapasztaltabb ***Bourne*** héj használók rámutatnának, hogy a parancshelyettesítést úgy is végre lehet hajtani, ha a parancsot ***fordított aposztrófok (back-tick)*** közé helyezzük. A ***Bourne***-stílusú parancshelyettesítéssel, a fenti dátum értékadás a következő alakot ölti:

```
today2=`date +%d-%b-%Y`
$ echo $today2
12-Mar-2004
```

Az újabb ***bash***-stílusú parancshelyettesítés formátumnak azonban van néhány fontos előnye. Először is könnyebben egymásba ágyazható. Mivel a nyitó és záró jelek különbözőek, a belső jeleket nem kell backslash-el védeni. Másodszor könnyebb olvasni, különösen ha beágyazva használjuk. Még ***Linuxon*** is, ahol a ***bash*** az általános, könnyen találkozhatunk a régi, ***Bourne*** stílusú formátumot használó héjprogramokkal. Ennek oka a hordozhatóság biztosítása a különféle ***UNIX*** verziók között, ahol nem mindig van elérhető ***bash*** viszont van ***Bourne*** héj. A ***bash*** visszafelé együttműködik a ***Bourne*** héjjal, így megérti a régebbi formátumot.

Szabványos hiba átirányítása

Előfordult már, hogy egy állományt kerestünk a `find` paranccsal, ám a keresett állomány eltűnt a `permission denied` hibaüzenetek tengerében melyek pillanatok alatt elözönlötték a terminálablakunkat?
Ha mi vagyunk a rendszergazdák, átjelentkezhetünk root felhasználóként, hogy úgy adjuk ki újra a `find` parancsot. Minthogy a root bármilyen állományt olvashat, többé nem kapunk hibaüzeneteket. Sajnos nem mindenki rendelkezik root jogokkal azon a rendszeren amit használ. Emellett meglehetősen rossz gyakorlat root-ként dolgozni hacsak nemfeltétlen szükséges. De vajon mi mást tehetünk? Az egyik megoldás, ha a kimenetet egy állományba irányítjuk. A szokásos alap kimenet átirányítás nem lehet újdonság annak aki bizonyos időt töltött ***UNIX*** vagy ***Linux*** héjkörnyezetben, így a kimenet átirányítás részleteibe most nem mennék bele. A `find` parancs hasznos kimenetének elmentéséhez a kimenetet a következőképpen irányíthatjuk egy állományba:

```
$ find /  -name foo > output.txt
```

A hibaüzeneteket továbbra is látni fogjuk a képernyőn, ám a keresett állomány elérési útját már nem. Ez ugyanis az `output.txt` állományba kerül. Amikor a `find` parancs befejeződik, kiírathatjuk a `output.txt` állomány tartalmát a `cat` paranccsal és máris láthatjuk a keresett fájl(ok) elérési helyét. Ez ugyan elfogadható módszer, de van jobb megoldás is. Ne a szabványos kimenetet irányítsuk át egy állományba, hanem a hibaüzeneteket. Ezt úgy érhetjük el, hogy közvetlenül az átirányító jel elé a 2-es számot írjuk. Ha nem érdekelnek bennünket a hibaüzenetek, nyugodtan elküldhetjük őket a */dev/null*-ba:

```
$ find /  -name foo 2> /dev/null
```

Így a bosszantó `permission denied` üzenetek nélkül nézhetjük meg a `foo` állomány elérési útját, feltéve persze, hogy létezik. Szinte mindig így hívom meg a `find` parancsot. A 2-es szám a szabványos hibacsatornát jelenti. A legtöbb program erre a szabványos hiba csatornára küldi a hibaüzeneteit. A szokásos (nem-hiba) kimenet a szabványos kimenetre kerül, amit egyébként az 1-es szám jelez. Minthogy a leggyakrabban átirányított csatorna a szabványos kimenet a kimenet átirányítás alapértelmezés szerint a szabványos kimenet folyamot irányítja át. Következésképpen a következő két parancs egyenértékű:

```
$ find / -name foo > output.txt
$ find / -name foo 1> output.txt
```

Előfordul, hogy a a hibaüzeneteket és a szabványos kimenetet is el szeretnénk menteni egy állományba. A `cron` folyamatok esetében gyakran van szükség ilyesmire, amikor minden kimenetet egy naplófájlba szeretnénk menteni. Ezt is megtehetjük, ha mindkét kimeneti folyamot egyazon állományba irányítjuk:

```
$ find / -name foo > output.txt 2> output.txt
```

Működik ugyan, de megint csak van egy jobb módszer. Az és jel segítségével a szabványos hibafolyamot a szabványos kimenethez köthetjük. Miután ezt megtettük, a hibaüzenetek ugyanoda mennek ahová a szabványos kimenetet irányítottuk:

```
$ find / -name foo > output.txt 2>&1
```

Egy dologra azonban oda kell figyelni, a kötés műveleti jele mindig a kimenetet készítő parancs végére kerül. Ez akkor

lehet fontos, ha a kimentet egy másik parancsba vezetjük át. A következő sor úgy működik ahogy várjuk:

```
find -name test.sh 2>&1 | tee /tmp/output2.txt
```

Ez viszont nem:

```
find -name test.sh | tee /tmp/output2.txt 2>&1
```

és a következő sem:

```
find -name test.sh 2>&1 > /tmp/output.txt
```

A kimenet átirányítás bemutatásához a `find` parancsot használtam példaként, majd valamennyi későbbi példa is ezt a parancsot használta. A megoldás azonban természetesen nem csak a `find` paranccsal működik. Rengeteg másik parancs is készít hibaüzeneteket amelyek elfedhetik az eredményül várt egy-két soros kimenetet.
A kimenet átirányítás sem ***bash*** különlegesség. Minden ***UNIX/Linux*** héj ugyanilyen alakban támogatja a kimenet átirányítást.

Keresés a parancstörténetben

A ***bash*** héj egyik legnagyszerűbb képessége a parancstörténet, melynek segítségével könnyedén navigálhatunk oda-vissza a már kiadott parancsok között a fel és lefelé nyilak segítségével. Mindez kiváló amíg csak az utóbbi 10-20 kiadott parancs között keresgélünk, de igen fárasztó, ha a parancs 75-100 parancsnyira van a parancstörténetben. A dolgok felgyorsítása érdekében interaktívan keresgélhetünk a parancstörténetben a ***Ctrl-R*** kombináció leütésével. Amint ezt megtettük a parancssor a következőre változik:

```
(reverse-i-search)`': 
```

Gépeljük be a keresett parancs néhány betűjét és a ***bash*** megmutatja melyik az a legfrissebb parancs amely tartalmazza az eddig begépelt betűket. Amit gépelünk, a ` és ‘ jelek között jelenik meg a parancssorban. Az alábbi példában, a `htt` szót gépeltem be:

```
(reverse-i-search)`htt': rpm -ql $(rpm -qa | grep
➥ httpd)
```

Ez azt jelenti, hogy a legutoljára begépelt parancs amely tartalmazza a `htt` karaktereket a:

```
rpm -ql $(rpm -qa | grep httpd)
```

amennyiben újra végre akarom hajtani a parancsot, egyszerűen csak Entert ütök. Ha inkább szerkeszteni szeretnénk, a jobb vagy bal nyíl segítségével ezt is megtehetem. Ilyenkor a parancssorban mutatott parancs a szokásos prompt sorba kerül, ahol ugyanúgy szerkeszthetem mintha begépeltem volna. Ez igazi időmegtakarítás lehet ha sok paraméterrel ellátott parancsot szeretnék a parancstörténet mélyéről előásni.

Ciklusok használata parancssorból

Az utolsó tipp amit be szeretnék mutatni, az, hogyan tudunk ciklusokat használni a parancssorban. A parancssor nem igazán az a hely ahol egymásba ágyazott ciklusokat és elágazásokat tartalmazó összetett szkripteket szokás írni. Kis ciklusok használatával azonban egész sok időt takaríthatunk meg. Sajnos nem sok emberrel találkoztam aki ki is használná ezeket a lehetőségeket. Inkább az a jellemző, hogy az emberek minden ciklusban visszalépkednek a parancstörténetben és módosítják az eredetileg beírt parancsot. Amennyiben valaki nem nagyon ismeri a `for` vagy egyéb ciklustípusok készítésének módozatait, érdemes utánaolvasni. Sok jó héjprogramozásról szóló könyv tárgyalja a kérdést. A `for` ciklusok általános tárgyalása önmagában is megérdemelne egy cikket.
Kétféleképpen lehet interaktív szkriptet írni. Az első, általam előnyben részesített módszer, ha minden sort pontosvesszővel választunk el. A könyvtárban található valamennyi fájlról háttérmentést készítő egyszerű ciklus a következőképpen nézne ki:

```
$ for file in * ; do cp $file $file.bak; done
```

A másik lehetőség, ha pontosvessző beszúrása helyett minden sor után Enter-t ütünk. A ***bash*** a `for` kulcsszóból felismeri hogy ciklust készítünk, és a másodlagos prompt alkalmazásával kéri be a következő sorokat. A `done` kulcsszóból meg tudja állapítani, hogy befejeztük a ciklust:

```
$ for file in *
> do cp $file $file.bak
> done
```

És most valami egészen más

Amikor eredetileg rászántam magam erre a cikkre, a ***„Bolondos bash trükkök”*** címet akartam adni neki, ahol bemutatok néhány különös, ezoterikus ***bash*** parancsot amelyeket megismertem. A cikk hangvétele azóta sokat változott, de egy bolond ***bash*** trükk azért megmaradt amit be szeretnék mutatni. Körülbelül öt évvel ezelőtt egy ***Linux*** rendszer, amelyért én feleltem, kifutott a memóriából. Még az olyan egyszerű parancsok is, mint az `ls` `insufficient memory` hibával álltak le. A probléma nyilvánvaló orvoslása egy egyszerű reboot lett volna. Az egyik rendszergazda azonban szeretett volna megnézni egy állományt amely esetleg a problémával kapcsolatos nyomokat tartalmazhat, de nem emlékezett a pontos fájlnévre. A könyvtárakba be tudtunk lépni, hiszen a `cd` parancs a ***bash*** része, de a fájllistát már nem tudtuk lekérdezni, hiszen még az `ls` sem működött. A probléma megkerülésére a másik rendszergazda készített egy egyszerű ciklust amely megmutatta a könyvtárban található állományokat:

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

Ez olyankor is működött amikor az `ls` nem, hiszen az `echo` a ***bash*** héj része, így már eleve a memóriába töltve található. Érdekes megoldása egy nem szokványos problémának. Meg tudja valaki mondani, hogyan lehet egy állomány tartalmát megjeleníteni kizárólag ***bash*** beépített parancsokat használva?

Összefoglalás

A ***bash*** héj rengeteg remek szolgáltatással könnyíti meg felhasználói életét. Remélem, legkedveltebb ***bash*** trükkjeim rövid összefoglalója mutatott néhány új lehetőséget, hogy jobban kihasználhassuk a ***bash*** igazi erejét.

Linux Journal 2005. április, 132. szám

Prentice Bisbal