Pingvin-fortélyok

Bevált trükkök Linux alatt mindennapjaink megkönnyítésére.

A napokban kért meg egy barátom, hogy telepítsek fel egy Debian Woody-t a számítógépére. Mivel csak most ismerkedik a Linuxszal, fontos volt, hogy az alaprendszer telepítésén túl azokba a mindennapi munkát segítő tippekbe is beavassam, amelyek hosszú kísérletezgetések eredményeként váltak tudásom részévé. Elmondta, hogy nem szeretné végigjárni ugyanezt az utat a varázslóvá válás érdekében. Álljon itt néhány titok azoknak, akik csak most fedezik fel azt az élményt, amit a Linux jelent. Csak a kezemet figyeljétek!

Top a 6-oson

Bizonyára te ismered a `top` nevű, remek segédprogramot, ami a */proc* állományrendszer adatait felhasználva listázza ki a legerőforrás-igényesebb folyamatokat. Ez a szép összesítő jól mutatna az egyik konzolon. A legkézenfekvőbb megoldás természetesen az, hogy belépünk a 6-oson és kiadjuk a `top` parancsot. Ennek számos hátránya van amellett, hogy nem szép megoldás: egyrészt újraindítás után újra el kellene kézzel indítani a `top`-ot. Másrészt, ha felállunk a számítógép mellől, egy nem túl szakavatott felhasználó is odaléphet, megnyomhatja a q-t (kilépés), és kap egy parancssort.
Felmerül a kérdés: nem lehetne-e a `top`-ot is ugyanúgy indítani, mint a */bin/login*-t? Nem tettem volna fel a kérdést, ha a válasz nem lenne egyértelmű: de igen! Kukkantsunk bele a */etc/inittab*-ba:

```
6:23:respawn:/sbin/getty 38400 tty6
```

Ez az `init` beállításállománya. Az `init` az első program, amelyik egy Linux-rendszeren elindul, ezért mindig az 1-es folyamatazonosítót kapja. Az ő feladata befűzni az állományrendszereket, elindítani a különböző démonokat és nem utolsó sorban előkészíteni a konzolokat a belépésre. A fent látható sor a 6-os konzolon teszi lehetővé a bejelentkezést. A : (kettősponttal) elválasztott mezők közül az első egy tetszőleges azonosító. A második mutatja, hogy mely futási szinteken kell elindítani. A harmadikban szereplő `respawn` utasítja az `init`-et arra, hogyha a program véget ér, indítsa újra (élessze fel). Az utolsó mező a számunkra is fontos rész, itt található ugyanis a `getty`. Ez az utóbbi egy terminált nyit meg és egy tetszőleges programot futtat rajta, alapértelmezésben a */bin/login*-t. Mi azonban arra kérjük, hogy inkább a `top` induljon el:

```
6:23:respawn:/sbin/getty -i -l /usr/bin/top
➥-n 38400 tty6
```

A `-i` kapcsolóval érhetjük el, hogy a `getty` ne írassa ki a */etc/issue* állomány tartalmát a `top` indítása előtt a képernyőre. A `-l` használata egyértelmű, míg a `-n` arra utasítja a programot, hogy ne kérjen felhasználói nevet. A változtatások akkor lépnek érvénybe, amikor az `init` újraolvassa az `inittab`-ot, és a 6-os konzol `getty`-je véget ér.

```
# kill -HUP 1
# ps aux | grep tty6
```

A második parancs kimenetéből böngészd ki a 6-os konzolhoz tartozó `getty` folyamatazonosítóját, és lődd ki a `kill` segítségével.
Ezzel azonban még nem vagyunk készen. A `top` ugyanis most többet tud, mint amennyit kellene. Ebből az alkalmazásból közvetlenül jelzéseket lehet küldeni a folyamatoknak. Ha így hagynánk ott a gépünket, a 6-os konzolon ez a felirat üvöltene: „Gyere és lődd ki a rendszergazda folyamatait a `top` segítségével!" Ennek elkerülésére a `top`-nak létezik egy biztonságos módja. Ha parancssorból egy `s` kapcsolóval indítod, nem fogja engedni a `kill` használatát. Ahhoz, hogy a `top` mindig biztonságos módban induljon el, globális beállításállományát, a */etc/toprc*-t kell szerkeszteni. Ha ez az állomány egy `s`-en és egy soremelésen kívül mást nem tartalmaz, már elégedettek lehetünk. Fut egy biztonságos `top` a 6-os konzolon, mely kilépés és újraindítás után is megmarad – ez volt a célunk.

Rendszergazda vagyok, de nem tudom a jelszót!

Biztosan te is hallottad már a nagy bölcseket, amikor óva intettek attól, hogy folyamatosan rendszergazdaként használd a gépet. Valóban nem tanácsos így tenni, mivel ekkor az összes általad indított folyamat is korlátlan jogosultságokkal bír. Csak a legszükségesebb esetben érdemes rendszergazdai parancssort indítani. A `su` pont erre való. Én legtöbbször a – (mínuszjellel) indítom, aminek hatására a kért felhasználó környezetében találjuk magunkat. A jelszavakkal azonban mindig gondban vagyok. Ha túl egyszerű, akkor más is rájöhet; ha túl bonyolult, sokáig tart begépelni. Nem lehetne, hogy az én mindennapi felhasználói fiókom varázsütésre rendszergazdaivá lépjen elő? Semmi sem lehetetlen, ha a

pingvin ereje veled van. A megoldást a PAM (Pluggable Authentication Modules) jelenti.
A PAM megjelenése választotta szét az azonosítást megkövetelő alkalmazásokat magától az azonosítás folyamatától. Egy olyan programnak, amely távoli bejelentkezést tesz lehetővé, nem kell foglalkoznia azzal, hogy a felhasználó hitelesítése jelszóval vagy ujjlenyomattal történik-e. Csupán megkérdezi a PAM-ot, hogy átengedje-e vagy sem. A PAM-ot pedig megfelelő jogosultságok birtokában mi, mezei rendszergazdák állíthatjuk be.
Régen a */etc/pam.conf* tartalmazta az összes szolgáltatás beállítását. Ma már ezek külön állományokban találhatók a */etc/pam.d* könyvtár alatt; minden fájl egy-egy szolgáltatáshoz tartozik. Számunkra most a */etc/pam.d/su* lesz érdekes:

```
# This allows root to su without passwords
➥(normal operation)
auth       sufficient pam_rootok.so
```

Ha a rendszergazdának lehet, nekünk miért nem? Lehet, csupán találnunk kell egy olyan modult, ami felhasználók egy csoportját tudja engedélyezni.

```
auth       sufficient pam_listfile.so onerr=fail
➥sense=allow file=/etc/su_ok item=ruser
```

A *pam_listfile.so* modul egy egyszerű szöveges állományt vesz alapul. Ebben minden sor egy-egy felhasználói név. Az `onerr` kapcsoló azt mondja meg, hogy a fájl megnyitásakor előforduló hiba esetén a `sense` által meghatározott hatás legyen-e érvényben. A mi beállításunk röviden annyit jelent, hogy ha a fájl nem található, akkor nem ért véget a hitelesítés, s ugrik a következő sorra (amely majd a *pam_unix.so* modullal a jelszót fogja kérni). A `sense` kapcsoló a hitelesítés sikeressége esetén kiváltandó hatást mondja meg. Ez azt teszi, hogyha a fájlban megtalálja a felhasználót, akkor engedélyezi a jelszó nélküli `su`-t.
A `file` kapcsolóval az állomány nevét adjuk meg, az `item` pedig a fájlban található sorok jelentését adja meg. Nemcsak a felhasználók neveit vehetnénk ide fel, hanem például csoportokat is, de a kettőt nem lehet keverni. A `ruser` a remote user (távoli felhasználó) rövidítése; azért távoli, mert nem azt a felhasználót keressük a fájlban, akinek a jogosultságait a `su`-val meg akarjuk szeretni, hanem azt, aki a `su` parancsot kiadta.
Végül mindössze egy */etc/su_ok* állományt kell létrehozni, és soronként felvenni azoknak a felhasználóknak a neveit, akiknek engedélyezni szeretnénk a jelszó nélküli belépést. Ezt az állományt – mondanom sem kell – nagyon jól kell védeni, ezért a létrehozás után az az első, hogy kiadjuk a `chmod 600 /etc/su_ok` parancsot.

Vipw, de nem vi

Az `EDITOR` környezeti változó tartalmazza az alapértelmezett szövegszerkesztő elérési útját. Sok alkalmazás használja ezt a változót, többek között a `vipw` is, amellyel a */etc/passwd* tartalma módosítható. Közvetlenül azért nem illik ezt az állományt szerkeszteni, mert a minden körültekintést mellőző felülírással a szerkesztés közben bejelentkezni próbáló felhasználók átmenetileg összeférhetetlenséget tapasztalhatnak. A `vipw` erre megoldást kínál, és az `EDITOR` által meghatározott szerkesztővel kényelmesen módosítható az állomány tartalma. Debian alatt az `EDITOR` alapértelmezésben nincs beállítva. A Bash-megfelelő héjak globális beállítási állománya a */etc/profile*. Ehhez a következő sort hozzáadva a legközelebbi belépéskor már minden felhasználónak a `joe` lesz az alapértelmezett szövegszerkesztője:

```
export EDITOR="/usr/bin/joe"
```

Ezek után a `vipw`-t kiadva már `joe`-ban dolgozhatunk.

Midnight kontra Norton

A jó öreg Norton Commanderen felnőtt felhasználók sokat panaszkodnak amiatt, hogy ha `mc`-ben egy bonyolult elérési úthoz lépnek, majd kilépnek a fájlkezelőből, nem ott találják magukat, ahová elbarangoltak, hanem ott, ahonnan indultak. Ez nagyon egyszerűen orvosolható: Debian alatt az `mc` csomag egy példaparancsfájlt tartalmaz, ezt felhasználva könnyedén megoldhatjuk a gondot. A */usr/lib/mc/bin/mc.sh*-ban a következő áll:

```
mc ()
{
        mkdir -p $HOME/.mc/tmp 2> /dev/null
        chmod 700 $HOME/.mc/tmp
        MC=$HOME/.mc/tmp/mc-$$
        /usr/bin/mc -P "$@" > "$MC"
        cd "`cat $MC`"
        rm -f "$MC"
        unset MC;
}

export -f mc
```

Ez egy `mc` nevű függvényt hoz létre, bizonyos értelemben felülírva az eredeti mc nevű futtatható állományt. Ez azt jelenti, hogyha a fenti parancsfájlt végrehajtjuk, akkor az `mc` parancsot kiadva nem érjük el közvetlenül az áhított programot, hanem ez a függvény fut le. Ami mindössze annyit tesz, hogy az `mc`-t a `-P` kapcsolóval indítja, s ilyen módon az futásának végeztekor visszaadja a legutolsó munkakönyvtárat, és így a `cd` paranccsal lépünk bele. Természetesen ezt a függvényt is minden egyes héj indításakor létre kell hozni, így a */etc/profile* állományba fel kell vennünk a következő sort:

```
. /usr/lib/mc/bin/mc.sh
```

A . (pont) ebben az esetben külső parancsfájl meghívását eredményezi.

Fülöp Balázs (admin@guardware.com)
18 éves, imádja a Túró Rudit, a Debian Linuxot és a teheneket. Kedvenc írója Slawomir Mrożek. Leginkább a számítógépes hálózatok biztonsága érdekli. A BME VIK műszaki informatikus szak hallgatója.