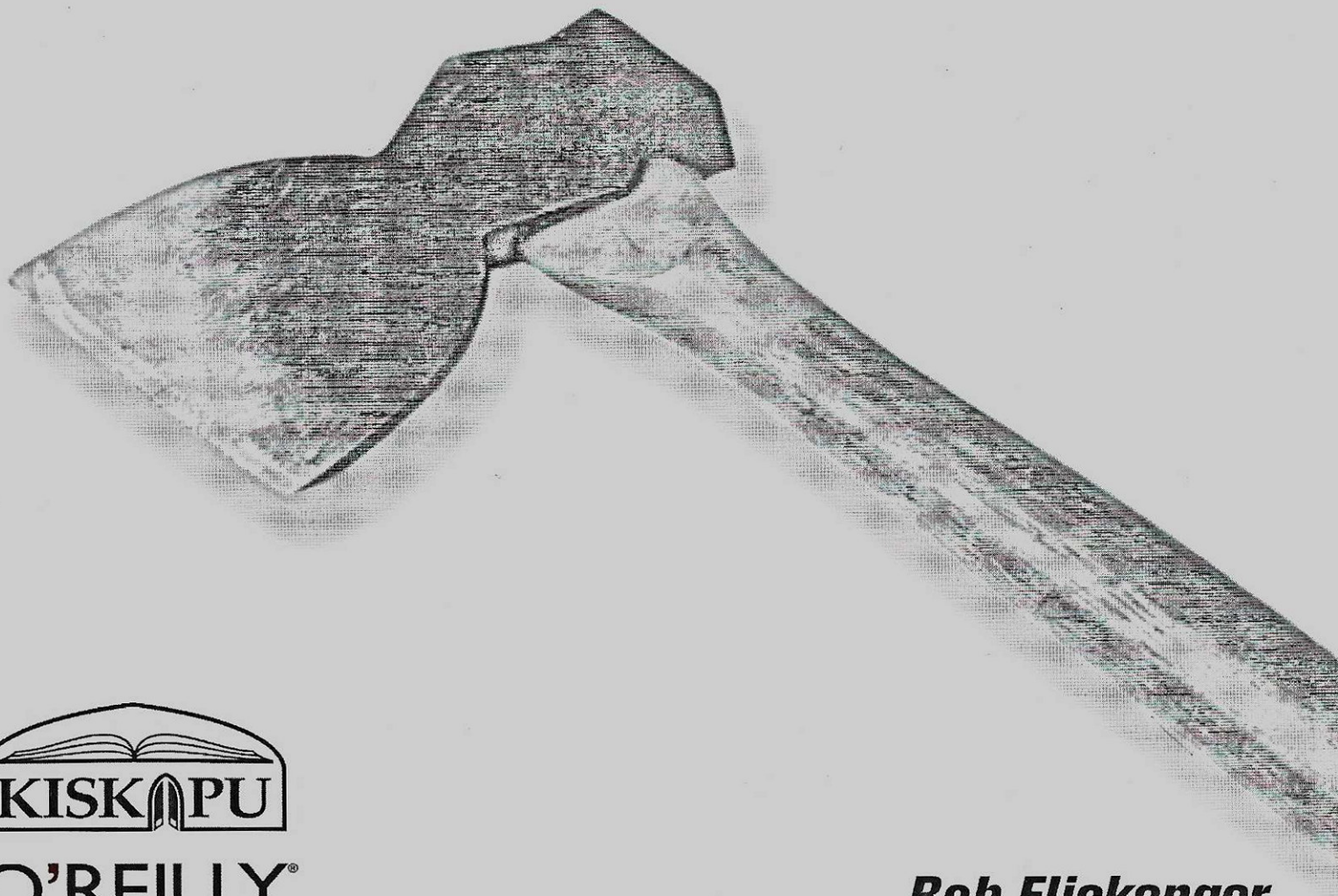
# LINUX BEVETÉS KÖZBEN



KISKAPU
O'REILLY®

*Rob Flickenger*

Rob Flickenger

# LINUX BEVETÉS KÖZBEN

Budapest, 2003
Kiskapu Kiadó

A kiadvány a következő angol eredeti alapján készült:

**Rob Flickenger: Linux Server Hacks**



**Felelős szerkesztő:** *Szy György*
**Nyelvi lektor:** *Nagy Anna*
**Szakmai lektor:** *Varga Csaba Sándor*
**Fordítás:** *Bagi Ádám, Barad Csaba, Mészáros Gergely, Nagy Zoltán*
**Műszaki szerkesztő:** *Csutak Hoffmann Levente*
**Tördelés:** *Csutak Hoffmann Levente*

Felelős kiadó a Kiskapu Kft. ügyvezető igazgatója
© 2003 Kiskapu Kft.
1081 Budapest, Népszínház u. 31., I. emelet, 7.
Telefon: (+36-1) 477-0443 Fax: (+36-1) 303-1619
http://kiado.kiskapu.hu/
e-mail: kiado@kiskapu.hu

**ISBN: 963 9301 55 8**

Készült a debreceni Kinizsi Nyomdában
Felelős vezető: *Bördős János*

# Tartalomjegyzék

# Előszó

## Hogyan lesz valakiből betyár?

A „Jargon File" tartalmaz egy csomó meghatározást az angol „hacker" szóra; legtöbbjük a műszaki hozzáértéssel, valamint a feladatmegoldás és a korlátok legyőzése felett érzett diadallal kapcsolatos. Magyar nyelvterületen hacker szót fordítják betyárnak, szakembernek, szakinak, vagy viccesen (az angol szó eredeti jelentéséből) csákányosnak. Annak, aki azt szeretné megtudni, hogyan *válhat* betyárrá, csak kettő fogalom érdekes igazán.

Létezik a nagy tudású programozóknak és hálózati szakembereknek az a közössége, közös kultúrája, melynek története az évtizedek során visszanyúlik az első időosztásos miniszámítógépekig és a legkorábbi ARPAnet kísérletekig. E kultúra résztvevőitől származik a „hacker" kifejezés. Ők építették ki az internetet és ők tették a Unix operációs rendszert azzá, amivé napjainkban lett. A szakemberek üzemeltetik a Usenetet és ők tartják működésben a webet. Azokat, akik részesei e kultúrának, fejlődéséhez hozzájárulnak, és a többiek ismerik, angol nyelvterületen hackernek, idehaza szakinak vagy betyárnak nevezik.

Ez a hozzáállás nem korlátozódik a programozók világára. Vannak, akik ezt a hozzáállást másra is kiterjesztik, például az elektronikára vagy a zenére – valójában ez a szemlélet megtalálható bármilyen tudomány vagy művészet magas szintű művelőinél.

Létezik egy másik csoport is, akik harsányan betyárnak, hackernek nevezik magukat, de nem azok. Ezeket az embereket (többnyire kamasz fiúk) lázba hozza, ha számítógépes rendszerekbe törhetnek be vagy megbéníthatják a telefonhálózatot. Őket betörőknek (cracker), illetve képzetlenebb társaikat buherátoroknak, vagy egyszerűen csak héjprogram-tacskóknak (script-kiddy) hívják. Az igazi szakemberek többnyire úgy gondolják, hogy a betörők (cracker) lusták, felelőtlenek és nem túlságosan értelmesek – csak azért, mert valaki képes feltörni egy rendszert, még nem lesz szakember, épp úgy, ahogy attól sem válik valaki közlekedésmérnökké, hogy össze tudja kötni a gyújtást egy autóban. Egy csomó újságírót és írót sajnos sikerült annyira megtéveszteni, hogy a betyár (hacker) kifejezést használják a betörőkre (cracker) – azonban ez mérhetetlenül bosszantja az igazi szakembereket.

Az alapvető különbség: a betyárok (hackers) létrehoznak, fejlesztenek valamit, a betörők (crackers) pedig csak rombolnak.

Aki betyárrá kíván válni, olvasson tovább. Aki betörő akar lenni, böngéssze az alt.2600 hírcsoportot, és készüljön föl rá, hogy öt–tíz évet rács mögött fog tölteni, miután kiderül, hogy nem olyan okos, mint amilyennek hitte magát. Ez minden, amit a betörőkről mondani szerettem volna.

## A betyárszemlélet

A szakemberek feladatokat oldanak meg és létrehoznak, valamint hisznek a szabadságban és a kölcsönös segítségnyújtásban. Ahhoz, hogy valakit betyárnak fogadjanak el, úgy is kell viselkednie. És ahhoz, hogy valaki úgy tudjon viselkedni, mint akinek ilyen a hozzáállása, tényleg hinnie kell ebben a szemléletben.

De aki azt gondolja, hogy a betyárszemlélet csak módszer arra, hogy a szakemberek társadalma befogadja, az elvéti a lényeget. *Neki* magának fontos, hogy olyan emberré váljon, aki hisz ezekben a dolgokban – ösztönzi és elősegíti, hogy új ismereteket szerezzen. Mint minden alkotó művészet esetében, a mesterré válás legjobb módja a mesterek szemléletének követése – nemcsak értelemmel, hanem érzelmekkel is.

Ahogyan a következő Zen vers mondja:

*Hogy kövesd az utat:*
*Figyeld a mestert,*
*Kövesd a mestert,*
*Kísérd a mestert,*
*Értsd meg a mestert,*
*Légy te a mester!*

Aki betyár akar lenni, annak addig kell ismételni a következő alapszabályokat, míg a vérébe nem ivódnak.

### 1. A világ tele van lenyűgöző, megoldásra váró feladványokkal

Betyárnak lenni nagyszerű szórakozás, de olyan szórakozás, ami rengeteg munkát igényel. A munkához pedig ösztönzés szükséges. A sikeres sportolók abból az élményből merítik erejüket, hogy a testüket nagyobb teljesítmény elérésére kényszerítik és legyőzik a saját fizikai korlátaikat. Ehhez hasonlóan, aki szakember akar lenni, annak ugyanilyen élményt kell jelentenie a feladatmegoldásnak, képességeinek csiszolásának és elméjének megdolgoztatásának.

Akinek ez nem a természetéből fakad, annak ilyenné kell válnia, hogy szakemberként megállja a helyét. Másképp a bütykölésre szánt energiáit olyan dolgok vonják el, mint a szex, a pénz és a társadalmi elismertség hajhászására.

(A szakembernek szüksége van arra is, hogy higgyen saját tanulási képességében – arra a hitre, hogy akkor is, amikor a meglévő tudása nem elég az adott feladat megoldásához, egy részlet feldolgozásából tanul annyit, ami a következő részlet megoldásához elég – és így tovább, amíg a végére nem jár.)

### 2. Egyetlen feladatot sem szabad kétszer megoldani

Az alkotóelme értékes és véges erőforrás. Nem szabad a kerék újra kitalálására pazarolni, amikor annyi izgalmas, új feladat vár megoldásra.

Aki betyárként akar élni, annak hinnie kell, hogy a többi betyár gondolkodásra fordított ideje értékes – olyannyira, hogy szinte erkölcsi kötelessége az ismeretek megosztása, a feladványok megoldása és a megoldások átadása, azért, hogy a többi betyár *új* feladatok megoldásával foglalkozhasson ahelyett, hogy újra és újra a régi nehézségekre tér vissza.

(Nem feltétlenül muszáj azonosulni azzal a gondolattal, hogy *minden* szellemi alkotást át kell adni, bár azoknak adóznak a többiek a legnagyobb tisztelettel, akik így tesznek. Természetesen összeegyeztethető a betyár (hacker) értékrendjével annyit eladni, amennyi a betevő falatra, szállásra és számítógépekre nélkülözhetetlen. Teljesen rendben lévő dolog a betyáros tulajdonságokat családfenntartásra, vagy akár meggazdagodásra használni, mindaddig, amíg az ember nem feledkezik el a hivatás és a többi betyár iránt érzett elkötelezettségéről.)

### 3. Az unalom és a robotolás bűnös dolog

A betyárokat (és a találékony embereket általában) soha nem lenne szabad unalomra vagy ostoba, ismétlődő feladatok elvégzésére kárhoztatni, mert ilyenkor nem azzal foglalkoznak, amihez értenek, vagyis nem az új feladványok megoldásán töprengenek. Ez a pazarlás mindenkinek kárt okoz. Ezért az unalom és a robotmunka nemcsak nyomasztó, de bűnös is.

Aki betyárként él, annak eléggé hinnie kell ebben ahhoz, hogy amennyire csak lehet, önműködővé teszi az unalmas részmunkákat, nemcsak saját magának, hanem mindenki más (és főleg a többi betyár) számára.

(Létezik ez alól egy elég nyilvánvaló kivétel is: a betyárok időnként cselekszenek olyasmit, ami a külső szemlélők számára ismétlődő vagy unalmas dolognak tűnhet, de csak összpontosítási gyakorlatként vagy olyan képességek, illetve tapasztalatok megszerzése érdekében

teszik, amit másképp nem lehetne elsajátítani. De ez önkéntes döntésen alapul – egyetlen értelmes lényt sem lenne szabad olyan helyzetbe kényszeríteni, ami untatja.)

### 4. A szabadság jó dolog

A betyárok természete szemben áll a hatalmaskodással. Aki parancsokat osztogathat, az megakadályozhatja, hogy a betyár annak a feladatnak a megoldásán dolgozzon, ami igazán izgatja – és a zsarnoki agyak már csak így működnek, erre többnyire valami taszítóan ostoba indokot találnak. Tehát a tekintélyelvű hozzáállás ellen mindenütt küzdeni kell, hogy ne taposhassa el a betyárokat.

(Ez nem egyenlő a mindenféle hatalom elleni küzdelemmel. A gyerekeket muszáj nevelni, a bűnözőket pedig féken kell tartani. A betyár is elfogad bizonyosfajta hatalmat, ha olyasmit kap, amire nagyobb szüksége van, mint az utasítások követésére fordított időre. De ez korlátozott mértékű, tudatos alku; arról a teljes behódolásról, amit a zsarnokok elvárnának, szó sem lehet.)

A hatalmaskodás a cenzúra és a titkolózás melegágya. A hatalmaskodók bizalmatlanok az önkéntes együttműködéssel és az ismeretek megosztásával szemben – csak azt az „együttműködést" kedvelik, amit ellenőrizhetnek. Ahhoz, hogy valaki betyárként viselkedhessen, ösztönös ellenérzést kell kifejlesztenie magában a cenzúra és a titkolózás minden fajtájával szemben, és a felelős felnőtt emberekkel szemben erővel vagy megtévesztéssel megvalósított kényszer ellen. Sőt, készen kell állniuk a cselekvésre is ennek érdekében.

### 5. A szemlélet nem helyettesíti a szakértelmet

Aki betyár akar lenni, annak ki kell alakítania ezt a szemléletet. De egy szemlélet utánzása még nem tesz senkit betyárrá, ahogyan bajnokká vagy rocksztárrá sem. Betyárrá lenni gondolkodás, gyakorlás, elszántság és kemény munka árán lehet.

Ezért bizalmatlannak kell lenni az öncélú stílussal szemben, és tisztelni kell a tudás minden fajtáját. A betyár nem vesztegeti az idejét a szakértelmet tettetőkre, de fejet hajt a tudás előtt – különösen a programozói tudás előtt, ugyanis a hozzáértés minden területen

megbecsülendő érték. Még jobb, ha kevesek által elsajátítható, kihívást jelentő dologhoz ért valaki, és a legjobb az éles elmét, szellemi erőt és összpontosítást igénylő területen szerzett szakértelem.

Aki tiszteli a szakértelmet, az szívesen fordít energiát a megszerzésére – így a kemény munka és az elkötelezettség inkább játékos tevékenység lesz, nem robotolás. Ez a hozzáállás nélkülözhetetlen a betyár számára.

A teljes esszé a `http://www.catb.org/~esr/faqs/hacker-howto.html` címen érhető el és az O'Reilly kiadó gondozásában megjelent „The Cathedral and the Bazaar" című könyv függelékeként olvasható.

Eric S. Raymond

*Eric S. Raymond a Jargon File alapján készült New Hacker's Dictionary és a híres „A katedrális és a bazár" esszé szerzője, amely a nyílt forrás mozgalom számára szolgált kovászként. Az Előszó szövege részlet az 1996-os „What is a hacker?" című írásából. Raymond szerint a betyárok az izgalmas feladatok megoldásában tűnnek ki, ez az O'Reilly „Hacks" sorozat alapgondolata is.*

# Bevezetés

A *hack* szónak számtalan jelentésárnyalata van, például: az „ügyes rögtönzés" az adott pillanatban, illetve a kihozza a legtöbbet a helyzetből az éppen kéznél lévő eszközök használatával. A csúnya „berhelés" a legátláthatatlanabb és legkevésbé érthető módon közelíti meg a kérdést, bár sok „jó trükk" is kibogarászhatatlannak tűnhet az avatatlanok szemében.

Egy adott rögtönzés hatékonysága elsősorban azon mérhető le, hogy mennyire alkalmas a feladat megoldására, és fordított arányban áll az alkalmazásához szükséges emberi beavatkozás mértékével. Egyes trükkök méretezhetőek, mások pedig akár hosszú távon is működtethetőek. A leghosszabb ideig működő és legáltalánosabban elfogadott rögtönzésekből szabványok lesznek, és számos más megoldás kitalálását indítják el. A jó trükk addig marad fenn, amíg jobbat nem találnak helyette.

A rögtönzés megmutatja hogyan illeszkedik egymáshoz a fejlesztő elvont és csodálatosan bonyolult elméje és az emberi igények tagadhatatlan és közönséges tapasztalata. Néha a trükk csúnya, és csak azért jött létre, mert valakinek viszketett valamije, és meg kellett vakarni. A mérnökök számára a barkácsolás a „csináld magad" szemlélet végső

kifejezése: senki nem érti jobban, hogyan jött létre a barkácsolás, mint az, aki úgy érezte, meg kell oldania az adott feladatot. Ha valaki, akinek a feladatmegoldás természetes beállítottsága, úgy érzi, hogy egy barkácsolás csúnya, akkor szinte biztosan ellenállhatatlan késztetést érez, hogy egy fokkal jobbat hozzon létre – és meghegeszti a barkácsolás végtermékét, amire e könyv olvasóit is bátorítani szeretnénk.

Végeredményben a legnagyobb teljesítményű kiszolgáló, a legtöbb memóriával és a világ legjobb (és legszabadabb) operációs rendszerével szintén csak egy csillogó-villogó hátvakaró, ami az adott pillanatban égető viszketést hivatott kezelni, amíg egy jobb, gyorsabb és olcsóbb hátvakaró le nem váltja.

Hová is vezet mindez az álfilozófikus szövegelés? Remélhetőleg rávilágít annak a szemléletnek a hátterére, ami elősegítette ennek a megoldásgyűjteménynek az összeállítását. Egyes megoldások rövidek és egyszerűek, mások meglehetősen összetettek. Mindegyik trükk egy adott feladat megoldására jött létre, amit a szerző nem tudott „vakarás" nélkül hagyni. Reményem szerint néhányat közvetlenül alkalmazhat mindenki egy-egy „viszketésre", amit linuxos kiszolgálók kezdő vagy tapasztalt üzemeltetőjeként érzett.

## A könyv elrendezése

A hozzáértő rendszergazda tudása rendkívül szerteágazó. Igazán hatékony csak az lehet, aki minden nehézséget képes megoldani, amivel a rendszer meglepi a bekapcsolástól a leállításig. Hogy a közbeeső idő átvészeléséhez segítséget nyújtsak, a következő csokrot gyűjtöttem össze a mindennapos rendszerfelügyeleti feladatok újszerű és időt megtakarító megközelítéseit.

- Az első fejezet mindenekelőtt azokat a leggyakoribb fajtába tartozó feladatokat vizsgálja meg, amelyekkel minden rendszergazda találkozik: beavatkozás a rendszerindítási folyamatba, hatékony munkavégzés a parancssorral, gyakori feladatok önműködővé tétele, az erőforrások felhasználásának megfigyelése (és szabályozása), és a Linux-rendszermag beállítása annak érdekében, hogy a rendszer hatékonyabban működjön. Mindez nem

bevezetés a rendszerfelügyeletbe, hanem olyan hatékony és nem kézenfekvő módszerek gyűjteménye, amelyeket még a tapasztaltabb rendszergazdák sem mindig vesznek észre.

- A *Változatkövetés* gyorstalpaló bevezetést nyújt két alapvető változatkövető rendszer, az RCS és a CVS használatába. Ha a rendszerbeállítások, a forráskód és a leírás tetszőleges korábbi változatát vissza tudjuk állítani, az néha megmentheti az állásunkat. Túl sok hivatásos rendszergazdának hiányosak az ismeretei a változatkövető rendszerekről (akik szívesebben hagyatkoznak inkább a nélkülözhetetlen, de kevésbé használható *.old* és *.orig* másolatokra). Ez a rész tömör és lényegre törő módon mutatja be a rendszer használatához szükséges parancsokat és ismereteket.
- A következő rész, a *Biztonsági mentések* gyors és egyszerű módszereket mutat be adataink biztonságba helyezésére. Külön hangsúlyt kap a hálózati mentés, az *rsync* és a CD-lenyomatok (CD-images) kezelése. Bemutatom a rendszer szabályos mentési eszközeiben rejlő óriási rugalmasságot, sőt rendszeres „pillanatfelvételekből" álló, a teljes fájlrendszer állapotát rögzítő mentési rendszer megvalósítására is mutatok egy módszert (amihez nincs szükség hatalmas tárhelyre).
- A könyvben a Hálózatkezelés (4. fejezet) a kedvencem. Nem az alapvető működtetésre és az útválasztásra összpontosít, hanem kevésbé ismert és őrületesen hasznos módszereket tárgyal, amikkel a hálózatot váratlan működési módokra bírhatjuk rá. Különféle IP-alagutakat állítunk be (titkosítással és anélkül), számba vesszük a NAT lehetőségeit és olyan haladó szolgáltatásokat ismerünk meg, amelyekkel mindenféle jellemzők függvényében érdekes eredményeket hozhatunk ki. Eszükbe jutott már, hogy a csomagok sorsáról a tartalmuk alapján döntsenek? Érdemes megnézni ezt a fejezetet.
- Az 5. fejezet olyan ötletek és eszközök eklektikus keveréke, amelyek segítenek kideríteni, miben sántikál éppen a kiszolgálónk. Sorra veszi a szabályos (és néhány teljesen nélkülözhetetlen „kiegészítő") csomagokat, amik rengeteget elárulnak arról, ki, mit, mikor és hogyan használ a hálózatunkon. Bemutat néhány módszert az elkerülhetetlen szolgáltatás-fennakadások enyhítésére is, és még annak a felderítésében is segíthet, amikor rossz szándékú emberek akarnak csínyeket elkövetni a hálózaton.

- Az *SSH* rész a titkosítással védett (és csodálatosan rugalmas) hálózati eszköz, az SSH mindenféle cseles felalkalmazási lehetőségét mutatja be. Az SSH számos változatban érhető el a Linuxhoz, ugyanakkor sok példa bár működik bármely változattal, az OpenSSH v3.4p1-es változat üzembiztosan működik, ezzel az összes példát kipróbáltuk.
- A 7. fejezet rövid kitérőt tesz a különféle megoldások bemutatásával, amiket egyszerűen nem lehet egyetlen parancssorba bepréselni. Ezek a tippek időt takarítanak meg, és remélhetőleg példát szolgáltatnak rá, hogyan lehet ügyes dolgokat összerakni a héj és a Perl használatával.
- A 8. fejezet a Linux három nagy alkalmazását mutatja be, ezek a *BIND 9*, *MySQL* és az *Apache*. Ez a rész feltételezi, hogy az olvasó bőven túljutott e csomagok egyszerű telepítésén, és olyan módszereket keres, amelyekkel gyorsabban és hatékonyabban lehet e szolgáltatásokat üzemeltetni anélkül, hogy sok munka lenne velük. Olyan módszereket ismerünk meg, amikkel gyorsan működésre bírhatjuk a kiszolgálót, nagyon nagy feladatokhoz is méretezhetjük, és olyan ügyes működési módokat hozhatunk ki belőle, amivel sok beállításra és karbantartásra fordított időt lehet megtakarítani.

## Hogyan használjuk ezt a könyvet

Hasznos lehet az is, ha valaki elejétől végéig elolvassa a könyvet, mivel az egyes megoldások valóban egy kicsit mindig a korábbiakra építenek. Ugyanakkor minden egyes rész külön is megállja a helyét, mint egy adott feladat egyik lehetséges megoldása. Ezért az egy témakörhöz tartozó megoldásokat szakaszokba vontam össze, de gyakran hivatkozom más szakaszba tartozó megoldásokra (és a témával kapcsolatos alapvető fontosságú forrásokra is). Az egyes szakaszokat nem szigorúan elhatárolt témakörökként kell elképzelni, inkább hasonló (de egymástól független) megoldások gyűjteményének. A legtöbben talán úgy fogják olvasni ezt a könyvet, mint a weboldalakat: megnéznek egy érdekes témát, és ha valamit nem értenek, követik a további forrásokra mutató hivatkozásokat.

## A könyvben alkalmazott jelölések

Az alábbiakban felsoroljuk könyvben használt tipográfiai jelöléseket:

*Dőlt betű*
: Új kifejezések, fájlnevek, fájlkiterjesztések, könyvtárak (directory), valamint programnevek.

`Állandó szélességű betű`
: Kódsorok, URL-ek, fájlok tartalma, parancsok és kapcsolók, értékek, modulok neve, valamint a parancsok kimenete.

**`Állandó szélességű félkövér betű`**
: Példákban és táblázatokban a betű szerint begépelendő parancsokat és szövegeket jelöli.

*`Állandó szélességű dőlt betű`*
: A példákban és táblázatokban azokat a szövegeket jelöli, amelyek a felhasználó által megadott értékekkel behelyettesíthetők.

## Visszajelzések

A könyv anyagát legjobb tudásunk szerint ellenőriztük, de kiderülhet, hogy időközben egyes programok megváltoztak (sőt még az is, hogy hibát vétettünk). Kérjük, jelezze nekünk a könyvben esetleg előforduló hibákat, pontatlanságokat, félrevezető vagy megtévesztő állításokat és elírásokat.

A következő címen érhet el minket:

kiado@kiskapu.hu
http://kiado.kiskapu.hu/

# A szerzőről

Rob Flickenger, a könyvben szereplő trükkök nagy részének szerzője. Rob a Slackware 3.5-ös változata óta dolgozik Linuxszal. Korábban az O'Reilly Network (tisztán linuxos környezet) rendszerfelügyeletét látta el, és ő a szerzője a *Building Wireless Community Networks* című könyvnek is, ami szintén az O'Reilly kiadó gondozásában jelent meg.

## Társszerzők

A következő emberek megoldásaikkal, írásaikkal vagy ötleteikkel járultak hozzá e könyv megszületéséhez:

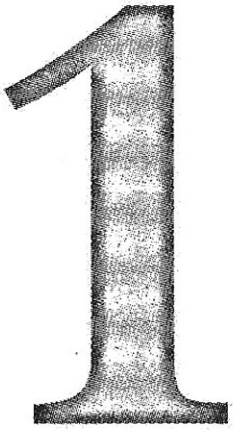
- Rael Dornfest ([#87] Apache Toolbox) az O'Reilly & Associates szakértője, aki leginkább a láthatáron éppen túl található műszaki megoldásokra összpontosítja figyelmét. Az O'Reilly Network és az O'Reilly Publications számára végez kutatást, kísérletezik, programozik és ír.
- Schuyler Erle (aki a httptop, a mysql-table-restore, a balance-push, a find-whois és a vtundgen kódhoz járult hozzá) nappal az O'Reilly & Associate szerény modorú internetes rendszerfejlesztője. Éjszaka az igazságosságért és a szabadságért küzd szabad programok írójaként és közösségi hálózatfejlesztőként.

- Kevin Hemenway ([#89] Gyors beállításváltás IfDefine használatával, [#90] Egyszerű, hivatkozás alapú hirdetéskövetés és a [#91] Utánozzunk Apache-val az FTP-kiszolgálókat), ismertebb nevén Morbus Iff, a disobey.com létrehozója. Több megoldást főzött ki és tett közzé, mint azt elképzelhetnénk, és mindenkinek szívesen kiosztana egy jókora serpenyő értelmet a feje búbjára, csupa szeretetből és illemtudóan.
- Seann Herdejurgen ([#47] IP Tables tippek és trükkök, [#62] Lemezelévülés-elemzés) 1987 óta dolgozik Unixszal, mostanában magas rendelkezésre állású megoldásokat épít vezető rendszermérnökként a D-Tech cégnél a texasi Dallasban. A Texas A&M Egyetemen szerzett MS fokozatot informatikából.
  A `http://seann.herdejurgen.com/` címen érhető el.
- Dru Lavigne ([#39] Biztonsági mentés a pax programmal) oktató egy magán műszaki főiskolán az oregoni Kingstonban. A TCP/IP-hálózatok, az útválasztás és a biztonság alapjait tanítja. Jelenlegi kedvtelési közé tartozik az összes TCP- és UDP-kapu társítása a hozzá rendelt alkalmazással és az összes RFC végigolvasása.
- Cricket Liu ([#77] Nézetek BIND 9 alatt) a Kaliforniai Berkeley Egyetemre jelentkezett, ami a szólásszabadság, a független Unix és az olcsó pizza nagyszerű védőbástyája. Egy évig a VeriSign Global Registry Services kötelékében dolgozott, a DNS-termékek irányítási igazgatójaként, és ő az egyik szerzője a *DNS and BIND* című, az O'Reilly & Associates kiadásában megjelent könyvnek.
- Mike Rubel ([#42], `http://www.mikerubel.org`) A Rutgers Egyetemen folytatott gépészmérnöki tanulmányokat (B. S. 1998.), a Caltechen pedig aeronautikát (M. S. 1999.) tanult, ahol jelenleg posztgraduális képzésben vesz részt. Évek óta szívesen használja a Linuxot és a GNU programokat kutatásai során.
- Jennifer Vesperman (a [#36] kivételével az összes CVS-tipp az O'ReillyNet számára készített cikkeiből származik) felhasználóként, íróként és esetenként programozóként járul hozzá a nyílt forrás világához. Programozói pályafutása során fejlődésében éppúgy szerepet játszik egy illesztőprogram megírása egy HDLC kártyához, mint Java grafikus programok ember–gép illesztőfelületeinek alakítása. Jenn a Linuxchix.org jelenlegi koordinátora és egyik rendszergazdája.

## Köszönetnyilvánítások

Szeretném megköszönni családomnak és barátaimnak a támogatást és bátorítást. Külön köszönet apának, amiért olyan kis koromban megmutatta a „helyes hibajavító módszert" és visszafordíthatatlanul elindított a betyárrá (hacker) válás útján, mielőtt még számítógépet láttam volna.

Természetesen ez a könyv mit sem érne azok csodálatosan tehetséges emberek munkája nélkül, akik hozzájárultak megoldásaikkal e kiadvány megszületéséhez. De természetesen mindannyiunk megoldásai óriások vállán barkácsolva jöttek létre (hogy ilyen borzasztó képzavarrá változtassam az ismert metaforát), és őszintén remélem, hogy az olvasó veszi az adást, vagyis amit itt tanult használja, sőt kitalál valami jobbat, és ami a legfontosabb, mindezt megosztja másokkal is.



# Kiszolgálóalapok

## 1–22. trükkök

A működő Linux-rendszer a gép alkatrészeinek és programjainak összetett együttműködése, ennek során láthatatlan démonok dolgoznak a felhasználó keze alá, szorgosan végezve rejtélyes feladataikat a megalkuvást nem ismerő feladatkezelő, a Linux-rendszermag dobjának ütemére.

A Linux-rendszer számtalan különböző feladat ellátására állítható be. Amikor munkaállomásként használjuk, a Linux látható fele az idejének nagy részét a grafikus felület kezelésével tölti: ablakokat rajzol a képernyőre és lesi a felhasználó minden jelzését és parancsát. Általában elvárjuk, hogy a gép nagyon rugalmas (és szórakoztató) legyen, amihez a jó válaszidő és az interaktivitás elsődleges fontosságú.

Ezzel szemben a Linux-kiszolgálókat néhány meghatározott feladat elvégzésére tervezik, ezekben szinte mindig szerepet kap nagy mennyiségű adat átpasszírozása valamilyen hálózati csatlakozáson – a lehető legrövidebb idő alatt. Egy csinos képernyővédő és a grafikus felület más szépségei meghatározók lehetnek egy sikeres irodai gép esetében, a Linux-kiszolgáló azonban olyan nagy teljesítményű eszköz, ami a lehető leggyorsabban és leghatékonyabban enged hozzáfé-

rést bizonyos adatokhoz. Az adatokat valamilyen tárolóeszközről (egy fájlrendszerből, adatbázisból vagy valahonnan a hálózatról) hívja be, és a hálózaton keresztül eljuttatja az adatokat kérő ügyfélhez, legyen az egy webkiszolgálóhoz csatlakozó ember, egy felhasználó a héjban, vagy esetleg egy másik kiszolgáló.

Ilyen körülmények között a rendszergazda (root) munkaköri felelőssége valahol a félisten és a gondnok bácsi szerepe között található. Végeredményben a rendszergazda feladata nem más, mint a lehető leggyorsabb (és legigazságosabb) módon hozzáférést adni a rendszer erőforrásaihoz. Ehhez a rendszergazdának tudnia kell kiépíteni új rendszereket (ezek akár már meglévő rendszerekre is ráépülhetnek), illetve képesnek (és hajlandónak) kell lennie rendet rakni olyan emberek után, akik anélkül használják a rendszert, hogy halvány lila fogalmuk lenne az „erőforrások felügyelete" kifejezés értelméről.

A legjobb rendszergazdák nem akadályozzák a rendszererőforrások használatát, hanem elősegítik, hogy a gépek elvégezzék a munkájukat. Felhasználóként akkor tudjuk, hogy a rendszergazda hatékonyan dolgozik, ha rendelkezésünkre állnak a munkánkhoz szükséges eszközök és soha semmit nem kell külön kérnünk tőle. Ezt a lehetetlennek hangzó feladatot a rendszergazda akkor oldhatja meg, ha előre látja a felhasználók igényeit és jól gazdálkodik a rendelkezésre álló erőforrásokkal.

Először is olyan módszereket mutatok be, amelyekkel a Linux hatékonyabbá tehető, hogy csak a szükséges feladatok ellátásával foglalkozzon, és ne vesztegessen processzoridőt olyan munkára, amire nincs is szükségünk. Példákat láthatunk arra, hogyan vehető rá a rendszer, hogy saját karbantartásának egyes feladatait átvegye, és olyan rejtett lehetőségeit aknázzuk ki, amelyek megkönnyítik munkánkat. A fejezet egyes részei (különösen a Parancssor és az Erőforrásfelügyelet szakasz) olyan módszereket mutatnak be, amelyeket nap mint nap használhatunk annak feltérképezésére, hogyan használjuk a rendszert és miként javíthatunk rajta.

Ezek a megoldások feltételezik, hogy az olvasó jártas a Linux használatában. A legfontosabb, hogy kísérletezni tudjon egy linuxos gépen,

amelyhez rendszergazdai jogosultsága is van, és megbízhatóan értsen a parancssor használatához. Nem árt, ha gyakorlott a hálózatok és az alapvető hálózati szolgáltatások kezelésében. Noha azt remélem, hogy a példák beszédesek lesznek, a linuxos rendszerfelügyeletbe való bevezetésül aligha jók. A helyes rendszerfelügyeleti módszereket részletesen tárgyaló munkák közül ajánlhatom az O'Reilly kiadónál megjelent Linux Network Administrator's Guide és Essential System Administration című könyveket.

A fejezetben található trükköket a következő öt csoportba soroltam: rendszerindítás, parancssor, önműködő feladatvégzés, erőforrásfelügyelet és a rendszermag finomhangolása.

*Rendszerindítás*

1. A kiszolgáló finomhangolása, hogy azt a szolgáltatást nyújtsa, amit valóban szeretnénk.
2. A bejelentkezés megkerülése.
3. Gyakran használt rendszermag-jellemzők.
4. Állandóan futó démon létrehozása az `init`-tel.

*Parancssor*

5. `n>&m`: a szabályos kimenet, valamint a szabályos hibacsatorna felcserélése.
6. Összetett parancssorok.
7. Cseles fájlnevek kezelése az *xargs* segítségével.
8. Megváltoztathatatlan állományok az `ext2`, illetve `ext3` fájlrendszerben.
9. A fordítások gyorsítása.

*Önműködő feladatvégzés*

10. Otthonos parancssori környezet.
11. A setuid és setgid programok kiküszöbölése.
12. Dolgozzon a `sudo`!
13. `Makefile` használata rendszergazdai feladatok önműködő végrehajtására.

*Erőforrásfelügyelet*

14. Tartománynév-keresés nyers erőből.
15. Ki zabálja fel a merevlemezt?
16. A */proc* szépségei.
17. Folyamatok közvetett kezelése a *procps* segítségével.
18. Rendszererőforrások felügyelete az egyes folyamatok szintjén.
19. Nagytakarítás a megszüntetett felhasználók után.

*A rendszermag finomhangolása*

20. A fölösleges meghajtók eltávolítása a rendszermagból.
21. Nagy mennyiségű RAM használata.
22. `hdparm`: az IDE-meghajtók finomhangolása.

# #1 A felesleges szolgáltatások eltávolítása

## A kiszolgáló finomhangolása, hogy azt a szolgáltatást nyújtsa, amit valóban szeretnénk

Ha kiszolgálót építünk, olyan rendszert szükséges létrehozunk, aminek a lehető leggyorsabban és leghatékonyabban kell ellátnia a feladatát, amire szánjuk. Ahogyan egy betonkeverő sem különösebben hasznos kiegészítője egy kávéautomatának, a feladattól idegen szolgáltatások erőforrásokat foglalnak le, és egyes esetekben komoly zavart okoznak, ami még csak nem is kapcsolódik a kiszolgáló eredetileg tervezett feladatához. Ez nem azt jelenti, hogy a Linuxot ne lehetne elsőrangú betonkeverőként foglalkoztatni, miközben még egy jó csésze kávét is készít, csak győződjünk meg róla, hogy éppen ezt akartuk-e elérni, mielőtt egy kiszolgálót rászabadítunk a világra (vagy inkább a világot a kiszolgálóra).

A kiszolgáló építése közben folyamatosan szem előtt kell tartanunk a kérdést: voltaképpen mire is akarom használni ezt a gépet? Valóban szükségem van-e FTP-szolgáltatásra a webkiszolgálómon? Fusson-e az NFS a névkiszolgálón, ha egyébként egyetlen állományrendszert sem osztok meg? Szükséges-e az `automount` démon, ha minden állományrendszert saját kezűleg fűzök be?

Egy egyszerű `ps ax` segítségével megtudhatjuk, miben sántikál gépünk. Ha nem lépett be más felhasználó, ez nagyjából megmutatja, hogy éppen mit futtat a kiszolgáló. Nézzük meg azt is, hogy az `inetd` milyen programok számára fogad kéréseket: használhatjuk a `grep -v ^# /etc/inetd.conf` vagy a (lényegre törőbb) `netstat -lp` parancsot. Az első utasítás azokat a sorokat mutatja az *inetd.conf* fájlból, amelyek nincsenek megjegyzésbe téve, a második pedig (rendszergazdaként kiadva) az összes foglalatot (socket), ami `LISTEN` állapotú, valamint az egyes foglalatokon figyelő programokat. Jó esetben a `ps ax` kimenetét le kellene tudnunk szorítani legfeljebb egy oldal hosszúságúra (leszámítva természetesen az olyan, önmagukat elágaztató kiszolgálóprogramokat, mint a `httpd`).

Íme néhány állandóan visszatérő (és többnyire teljesen felesleges) szolgáltatás, amit számos Linux-változat alapértelmezés szerint elindít:

`portmap`, `rpc.mountd`, `rpc.nfsd`
: Ezek egytől egyig az NFS rendszer részei. Futtatunk egyáltalán NFS-kiszolgálót? Szükségünk van távoli NFS-fájlrendszerek elérésére? Amennyiben egyik kérdésre sem válaszoltunk igennel, nincs szükségünk ezekre a démonokra. Szabadítsuk föl az erőforrásokat, amit lekötnek, és szabaduljunk meg egy esetleges biztonsági kockázattól.

`smbd` és `nmbd`
: Ezek a Samba démonok. Fogunk-e SMB alapú fájlmegosztást végezni windowsos (vagy más) gépek számára? Ha nem, akkor nyugodtan kilőhetjük ezeket a folyamatokat.

`automount`
: Az `automount` démon jó szolgálatot tehet a hálózati (vagy helyi) meghajtók igény szerinti befűzésével, lehetővé téve a hozzáférést rendszergazdai jogosultságok nélkül is. Ez különösen kapóra jön a munkaállomásokon, ahol a felhasználók cserélhető adathordozókat (CD-ket és hajlékonylemezeket) használnak, vagy hálózati erőforrásokhoz kell hozzáférniük. A legtöbb egy feladatra szánt kiszolgálón azonban az `automount` valószínűleg fölösleges. Ha a kiszolgáló nem tesz elérhetővé konzolos hozzá-

férést vagy megosztott hálózati meghajtókat, leállíthatjuk az `automount`-ot (és beállíthatjuk a fájlrendszerek statikus befűzését a */etc/fstab* állományban).

`named`

Üzemeltetünk-e névkiszolgálót? Szükségtelen a `named` futtatása, ha csupán hálózati neveket kell feloldani; ugyanis erre valók a */etc/resolv.conf* és a *bind* programkönyvtárak. Ha a gép nem nyújt DNS-szolgáltatást más gépeknek, és nem futtatunk csak gyorstárazó névkiszolgálót, akkor a `named` sem kell.

`lpd`

Nyomtatunk-e valaha erről a gépről? Amennyiben internetes erőforrásokat szolgáltat, jó eséllyel amúgy sem szeretnénk, ha nyomtatási kéréseket fogadna. Ha úgysem fogjuk használni, távolítsuk el a nyomtatódémont is.

`inetd`

Valóban szükség van rá, hogy valamelyik szolgáltatás az `inetd` felügyelete alatt fusson? Ha az `ssh` önálló démonként fut, és csak önálló démonokat használunk (mint például az Apache, a BIND, a MySQL vagy a ProFTPD), akkor az `inetd` fölösleges lehet. A legkevesebb, amit megtehetünk, hogy ellenőrizzük a `grep` paranccsal, milyen kéréseket fogad el az `inetd`: `grep -v ^# /etc/inetd.conf`.
Ha úgy találjuk, hogy minden szolgáltatást nyugodtan megjegyzésbe lehet tenni, akkor miért fusson a démon? Távolítsuk el a rendszerindítási folyamatból (akár úgy, hogy eltávolítjuk a rendszer *rc* állományaiból, akár egy egyszerű `chmod -x /usr/sbin/inetd` paranccsal – a szabályos megoldás, ha minden rc.[1-5] könyvtárból kifűzzük.).

`telnet`, `rlogin`, `rexec`, `ftp`

A távoli belépés, programvégrehajtás és fájlátvitel lehetőségét, amit ezek az ősrégi démonok nyújtanak nagyrészt kiváltja az `ssh` és az `scp`, a fenti programok titkosítással biztosított és hihetetlenül sokoldalú megfelelői. Ha csak nincs valami *nagyon* jó okunk rá, hogy valamelyiket megtartsuk, tanácsos eltávolítani őket a gépünkről. Ha nagyon szükségünk van az FTP-kapcsolat

támogatására, akkor próbát tehetünk a `mod_sql` bővítménnyel (plugin) a `proftpd` programhoz (lásd [#85] „Használjunk `proftpd` azonosítási forrásként MySQL-t").

`finger`, `comsat`, `chargen`, `echo`, `identd`

A `finger` és `comsat` szolgáltatást azokban az időkben volt értelme használni, amikor az internetfelhasználók még kíváncsi, de alapjában véve jóindulatú emberek voltak. Manapság a titkos pásztázások és túlcsordulási hibákra épített hálózati támadások korában olyan többletszolgáltatásokat működtetni, amelyek a kiszolgálóról árulnak el részleteket, alapjában véve *rossz* ötlet.

A `chargen` és az `echo` kapu (port) valamikor jól használható eszköz volt a hálózati kapcsolat ellenőrzésére, mára azonban túlságosan csábító célponttá vált a csínytevők szemében (akik talán még egymáshoz is kapcsolják a kettőt, hogy gyorsan és takarékosan feltornásszák a kiszolgáló terhelését).

Az `identd` valaha értelmes adatok fontos forrása volt, amely a kiszolgálóknak árult el arról valamit, hogy miféle felhasználók csatlakoznak a géphez. Manapság, amikor sajnos akármelyik helyi felhasználó rendszergazdai jogosultságokat szerezhet a biztonsági rések kihasználásával, és a Linuxot személyi számítógépeken is használják, gyakran előfordul, hogy az `identd`-t tulajdonképpen (minő borzalom!) azért telepítik, hogy *hamis* adatokat küldjön a gépről, bár a legtöbb kiszolgáló amúgy is figyelmen kívül hagyja ezeket az adatokat. Mivel az `identd` ennyire megbízhatatlan adatforrás, miért engedélyeznénk a futását?

A felesleges szolgáltatások kiküszöbölése érdekében először állítsuk le őket (vagy olyan módon, hogy kiadjuk a `szolgaltatasnev stop` parancsot a */etc/rc.d/init.d/* könyvtárban és eltávolítjuk a */etc/inetd.conf/* állományból, vagy egyenként a `kill` paranccsal). Majd, hogy a következő újraindításkor se indulhassanak el, távolítsuk el a szolgáltatásokat a */etc/rc.d/** parancsfájlokból. Amikor kigyomláltuk a felesleges szolgáltatásokat, ellenőrzésképpen indítsuk újra a gépet és ismét nézzük meg a futó folyamatokat.

Ha mindenképpen szükségünk van kétes biztonságú szolgáltatások futtatására, használjunk TCP-burkolót (wrapper) vagy helyi tűzfalat,

és korlátozzuk a hozzáférést azokra a gépekre, amelyeknek feltétlenül szükséges az adott szolgáltatás.

### Kapcsolódó anyagok

- [#45] Tűzfal létrehozása egy tetszőleges kiszolgálógép parancssorából
- [#78] Készítsünk gyorstárazó DNS-t a helyi tartományokhoz
- [#85] Használjunk *proftpd* azonosítási forrásként MySQL-t

## #2 A bejelentkezés megkerülése

### Teljes hozzáférés, semmi jelszó

Előbb vagy utóbb mindenkivel megtörténik: egy barátunk vagy ügyfelünk elfelejti a rendszergazdai jelszót, és olyan gépen kell dolgoznunk, amihez nincs is hozzáférésünk.

Ha nem csak hálózaton keresztül nyílik lehetőségünk a belépésre, és azt sem bánjuk, hogy újra kell indítani a gépet – a közismert bölcs tanács szerint – egyfelhasználós módban próbáljuk meg elindítani. Miután megnyomtuk a CONTROL-ALT-DELETE billentyűkombinációt (igény szerint a háromgombos újraindítást letilthatjuk a `/etc/inittab` fájlban – Lektor), egyszerűen várjuk meg az indítás utáni tesztet, és a `single` kapcsolóval indítsuk a rendszert. Például a LILO parancssorából ilyen módon:

```
LILO: linux single
```

Sok gép ezután vidáman ad egy rendszergazdai parancssort. De bizonyos rendszereken (például Red Hat, de ezt mi magunk is beállíthatjuk a `lilo.conf` fájlban a `restricted` és `passwd` kapcsolókkal. Ekkor a fájlt is védjük le a `chmod 600 lilo.conf` parancs segítségével – Lektor) a hírhedt biztonsági üzenet fogad:

```
Give root password for maintenance
(or type Control-D for normal startup)
```

Ha tudnánk, mi a rendszergazdai jelszó, nem lennénk itt! Ha szerencsénk van, az `init` parancsfájl ezen a ponton megengedi, hogy egyszerűen a ^C leütésével rendszergazdai parancssorhoz jussunk.
A legtöbb `init` folyamat azonban „okosabb" ennél és elfogja a ^C-t.

Mit tegyünk ilyenkor? Bármikor indíthatjuk a gépet lemezről is, de mi történik akkor, ha nincs kéznél rendszerindító lemez (boot disk), vagy nincs a gépben CD-ROM-meghajtó?

Nincs még veszve minden. Hogy elkerüljük az imént bemutatott kavarodást, módosítsuk a rendszert különös kegyetlenséggel, rögtön az indításnál. A LILO parancssorban adjuk ki:

```
LILO: linux init=/bin/bash
```

Mi lesz az eredménye? Ahelyett, hogy a */sbin/init* indulna el, és a szokásos */etc/rc.d/** indítási eljárás zajlana le, egyszerűen az mondjuk a rendszermagnak, hogy adjon egy parancshéjat. Nincs jelszó, nincs fájlrendszer-ellenőrzés (sőt, szinte semmi felhasználói környezet), de van egy jó gyors, csillogó-villogó parancssorunk.

Ez még sajnos nem igazán elég a javítási munkálatokhoz, ugyanis a fő állományrendszer csak olvasható módban lesz befűzve (mivel esélyt sem kapott rá, hogy az ellenőrzés után írható-olvasható módban újra befűzzék). Ezenkívül a hálózat sem érhető el és a szokásos rendszerdémonok sem futnak. Egy ilyen parancssort nem szokás a jelszó visszaállításánál (esetleg egy-két állomány megpiszkálásánál) többre használni. Mindenekelőtt ne lépjünk ki ^D-vel a héjból! Kicsiny héjunk (és a rendszermag) alkotja ugyanis ebben a pillanatban az egész Linux-rendszert. Tehát hogyan férhetünk hozzá a fájlrendszerhez, ha az csak olvasható? Próbáljuk meg a következőképpen:

```
# mount -o remount,rw /
```

Ez a fő állományrendszert írható-olvasható módban fűzi be újra. Most már megváltoztatható a rendszergazdai jelszó a `passwd` programmal (ha a rendszergazda egyszer már elfelejtette a jelszót, gondoljuk meg, hogy milyen következményekkel járhat, ha újra megadjuk neki. Aki pedig maga felelős a jelszó elvesztéséért, jól fontolja meg, nem kelleme-e felírnia láthatatlan tintával egy ragasztós jegyzetcédulára és föl-ragasztania a képernyőre vagy a fehérneműjébe, vagy esetleg nem lenne-e itt az ideje más kedvtelés után néznie).

Ha megadtuk az új jelszót, *ne indítsuk újra* a gépet. Mivel nem fut az `init`, nincs semmi, ami biztonságosan leállíthatná a rendszert. A biztonságos leállítás leggyorsabb módja, ha csak olvashatóként újrafűzzük a fő fájlrendszert:

```
# mount -o remount,ro /
```

A csak olvasható fő állományrendszer esetén nyugodtan megnyomhatjuk a RESET gombot, egyfelhasználós módban indíthatjuk a rendszert (vagy ehelyett a szükséges futásszintre lépünk, majd felépíttetjük az egész fájlrendszert: `init 2; mount -a` – Lektor) és nekiveselkedhetünk a tulajdonképpeni munkánknak.

## #3 Gyakran használt rendszermagkapcsolók

### Indításkor adjunk meg magkapcsolókat

Mint azt a [#2] „A bejelentkezés megkerülése" részben láthattuk, megadhatunk rendszermag-kapcsolókat a LILO parancssorában, és ezzel megválaszthatjuk a programot, amit a rendszer indításakor először hív meg. Az `init` módosítása (az `init=/bin/bash` sorral) csak egyike a számos hasznos beállításnak, ami rendszerindításkor megadható. Következzenek a leggyakoribb kapcsolók:

`single`
: Egyfelhasználós módban indítja a rendszert.

`root=`
: Megváltoztatja a / alá befűzött állományrendszert. Például a `root=/dev/sdc4` a harmadik SCSI-lemez negyedik lemezrészéről indítja el a rendszert (a rendszerbetöltő program alapértelmezett beállítása helyett).

`hdX=`
: Módosítja az IDE-merevlemezek adatait. Ez akkor hasznos, ha a BIOS helytelen adatokat ad:
`hda=3649,255,63 hdd=cdrom`
Eszerint az elsődleges (master) IDE-meghajtó 30 GB-os, LBA-módban működő merevlemez, a másodlagos (slave) meghajtó pedig a CD-ROM.

`console=`
: Soros kapuhoz csatlakozó konzolt határoz meg a soros konzolokat támogató rendszermagok számára. Például a `console=ttyS0,19200n81` arra utasítja a rendszermagot, hogy a rendszerindítási üzeneteket a `ttyS0` kapura (az első soros kapura) küldje, 19 000 baud sebességgel, paritásbit nélkül, 8 adat- és 1 stopbittel. Ne feledjük, ahhoz, hogy valódi soros konzolt kapjunk (amin be is lehet jelentkezni), egy ehhez hasonló sort kell adnunk a */etc/inittab* fájlhoz:
```
s1:12345:respawn:/sbin/agetty 19200 ttyS0 vt100
```

`nosmp`
: Letiltja a többprocesszoros működést az erre alkalmas rendszermagban. Ez akkor segíthet rajtunk, ha a rendszermaggal kapcsolatos hibára gyanakszunk egy többprocesszoros gépnél.

`mem=`
: Megadja a teljes rendelkezésre álló rendszermemória méretét. Lásd a [#21] „Nagy mennyiségű RAM használata".

`ro`
: A / lemezrészt csak olvasható módban fűzi be (általában ez az alapértelmezés, és csak az `fsck` futtatása után fűzzük újra írható-olvasható módban).

`rw`
: Írható-olvasható módban befűzi a / lemezrészt. Ez általában nem túl jó ötlet, ha nem barkácsolunk az `init`-tel. Megadhatjuk az `init` kapcsolót az `rw`-vel együtt `init=/bin/bash rw`, ezzel szükségtelenné téve azt a buta `# mount -o remount,rw /` dolgot a [#2] „Bejelentkezés megkerülése" részben.

Az előzőekben említetteken kívül átadhatunk kapcsolókat a SCSI-vezérlőnek, az IDE-eszközöknek, a hangkártyának és szinte minden egyéb eszközmeghajtónak. Minden meghajtó különböző, és jellemzően a megszakítások, kezdőcímek, paritás, sebesség, önműködő felismerés és hasonlók beállítását teszi lehetővé. A kimerítő részletek az elektronikus leírásokban megtalálhatóak.

### Kapcsolódó anyagok

- `man bootparam`
- */usr/src/linux/Documentatioin/**

## #4 Állandóan futó démon létrehozása az inittel

### Teremtsük meg a feltételeit, hogy a folyamat újrainduljon, bármi történjék!

Sokféle parancsállománnyal megoldható, hogy egy folyamat újrainduljon, ha váratlanul leállna. A legegyszerűbb valószínűleg valami ilyesmi lenne:

```
$ while : ; do echo "Run some code here...";
➥ sleep 1; done
```

Ha az `echo` helyett egy nem háttérben futó folyamatot indítunk, biztosítjuk, hogy mindig fusson (vagy legalábbis *megpróbáljon* futni). A : (kettőspont) egyszerűen mindig igaz feltételt ad a `while` ciklusnak (és hatékonyabb, mint a */bin/true* használata, mert nem szükséges minden egyes alkalommal külső programot meghívnia). Semmiképpen *ne* indítsunk háttérben futó folyamatot az `echo` helyén, ha nem találjuk szórakoztatónak, hogy színültig telik a folyamatleíró tábla (ugyanis a `while` mindaddig miden másodpercben indít egy példányt a megadott programból, amíg csak tud). Ha az ügyes trükkökkel vetjük össze, ez a `while`-megoldás nyilvánvalóan szerény használati értékű.

Mi történik, ha a program egyik futási feltétele nem teljesül? Azonnal kilép, azután minden másodpercben újra próbálkozik, anélkül, hogy valami jelzést adna a hibáról (hacsak a program nem használ saját naplózó rendszert vagy a `syslog` démont). Érdemes lenne figyelni a folyamatot és leállítani a próbálkozásokat, ha néhány próba után túl gyorsan ismétlődnek.

Minden Linux-rendszer tartalmaz egy olyan segédprogramot, amely önműködően elvégzi ezt a feladatot; ez az `init`. Az a program, ame-

lyik a rendszerindítást és a terminálok felélesztését végzi, tökéletesen alkalmas rá, hogy biztosítsa a programok állandó futását. Igazából ez az elsődleges feladata.

Tetszőlegesen adhatunk új sorokat a */etc/inittab* állományhoz olyan programokkal, amiket az `init` fog figyelni:

```
zz:12345:respawn:/usr/local/sbin/my_daemon
```

Az `inittab` bejegyzések kétbetűs, tetszőleges (de egyedi) azonosítóval kezdődnek (ebben az esetben zz), amit azoknak a futási szinteknek a száma követ, amelyekben a programot futtatni kell, majd a `respawn` kulcsszó, végül pedig a program teljes elérési útvonala. A fenti példában, amennyiben `az_en_demonom` olyan módon van beállítva, hogy az előtérben fusson, az `init` mindig elindít egy új példányt, valahányszor a program kilép. Az `inittab` módosítása után ne felejtsünk el HUP jelet küldeni az `init`-nek, hogy újratöltse a beállításait. Ennek gyors módja:

```
# kill -HUP 1
```

E parancs elegánsabb megadása az `init q`. Ha a program túl gyorsan indul újra, az `init` egy időre felfüggeszti a végrehajtást, hogy ne foglaljon le túl sok erőforrást. Például:

```
zz:12345:respawn:/bin/touch /tmp/timestamp
```

Ennek hatására a */tmp/timestamp* állomány módosítási idejét másodpercenként jó néhányszor átírja a *touch* program, egészen addig, amíg az `init` úgy nem dönt, hogy ami sok, az sok. Szinte azonnal a következő üzenetet láthatjuk a */var/log/messages* állományban:

```
Sep Sep 8 11:28:23 catlin init: Id "zz" respawning
➥ too fast: disabled for 5 minutes
```

Öt perc elteltével az `init` újra megpróbálja futtatni a parancsot, és ha még mindig túl sűrűn indul újra, akkor megint letiltja.

Ez a módszer nyilvánvalóan jól használható, ha a program rendszergazdai jogosultságokkal fut, de mi a helyzet akkor, ha más felhasználó nevében akarunk önműködően újrainduló programot futtatni? Ez sem jelent gondot a `sudo` használatával:

```
zz:12345:respawn:/usr/bin/sudo -u rob /bin/touch
➥ /tmp/timestamp
```

Most a *touch* rob felhasználói néven fut, nem rendszergazdai jogosultsággal. Aki olvasás közben ki is próbálja ezeket a parancsokat, feltétlenül távolítsa el a meglévő */tmp/timestamp* állományt, mielőtt a `sudo` sorral próbálkozna. Miután egy HUP jelet küldtünk az `init` démonnak, nézzük meg a *timestamp* állományt:

```
rob@catlin:~# ls -al /tmp/timestamp
-rw-r--r-- 1 rob users 0 Sep 8 11:28 /tmp/timestamp
```

Ha az `init` segítségével futtatunk egy démont két hátránya is akad: az egyik az, hogy az *inittab* fájlban megjegyzésbe kell tennünk a megfelelő sort hogyha le akarjuk állítani a démont (mert mindig újraindulna, amikor a `kill` leállítja), a másik pedig az, hogy csak a rendszergazda adhat új bejegyzéseket az *inittab* állományhoz. Ezzel együtt, ha arra van szükség, hogy egy folyamat mindig fusson bármi történjen is, az `init` nagyszerű munkát végez.

### Kapcsolódó anyag

- [#12] A munka neheze: a `sudo`

## #5 n>&m: a szabályos kimenet és a szabályos hibacsatorna felcserélése

### Irányítsuk a szabályos kimenetet és a szabályos hibacsatornát, ahova csak akarjuk!

Alapértelmezés szerint a programok hibaüzenetei a terminálra kerülnek vagy valahová (fájlba, csővezetékbe vagy fordított egyszeres idézőjelek közé) átirányítjuk őket.

Néha épp az ellenkezőjére van szükségünk. Például arra, hogy egy program kimenete a képernyőre kerüljön, a hibaüzeneteket (szabályos hibacsatorna) pedig fordított egyszeres idézőjelekkel szeretnénk elfogni. Esetleg a kimenetet egy fájlba akarjuk küldeni, a hibacsatornát pedig egy csővezetéken keresztül egy hibakezelő programhoz. A Bourne héjban ezt a következőképpen tehetjük meg (a C héjban mindez nem lehetséges).

A 0, 1 és 2 fájlleíró (inode) a szabályos bemenetet, szabályos kimenetet és a szabályos hibacsatornát jelöli, ebben a sorrendben. Átirányítás nélkül mindegyik a */dev/tty* terminálfájlhoz kapcsolódik. Bármelyiket egyszerűen átirányíthatjuk tetszőleges állományba (ha ismert a fájl neve). Például a hibacsatorna átirányításához egy *hibafajl* nevű állományba a következő utasítást használjuk:

```
$ parancs 2> hibafájl
```

A szabályos kimenetet csővezeték karakterrel és fordított egyszeres idézőjelekkel is átirányíthatjuk:

```
$ parancs | ...\
$ var=`parancs`
```

De a csővezetékhez és a fordított egyszeres idézőjelekhez (aposztróf) nem társul fájlnév, így nem használható a 2> átirányítás. Át kell rendezni a fájlleírókat anélkül, hogy ismernénk a fájlnevet (vagy valamit), ami hozzájuk kapcsolódik. Elmondom hogyan.

Haladjunk lépésről lépésre: először is küldjük a szabályos kimenetet és a hibacsatornát a csővezetékbe vagy a fordított egyszeres idézőjelekbe. A Bourne héj n>&m műveleti jele (operator) átrendezi a fájlneveket és a fájlleírókat. Jelentése: „mutasson az *n* fájlleíró ugyanoda, ahová az *m*". Alkalmazzuk ezt az előző példára, ugyanoda küldjük a hibaüzenetet, ahová a szabályos kimenet megy:

```
$ parancs 2>&1 | ...
$ var=`parancs 2>&1`
```

A 2>&1 mindkét példában azt jelenti, hogy „küldd a szabályos hibaüzenetet (2 fájlleíró) oda, ahová a szabályos kimenet (1 fájlleíró) megy". Egyszerű, nem?

Több n>&m műveleti jel is használható. A héj mielőtt végrehajtaná a parancsot, balról jobbra haladva, értelmezi őket.

„Aha!" mondhatnánk, „ahhoz, hogy felcseréljük a szabályos kimenetet a szabályos hibacsatornával – azaz a stderr a csővezetékbe, a stdout pedig a képernyőre kerüljön – csupán ennyi szükséges":

```
$ parancs 2>&1 1>&2 | ... (hibás...)
```

Nem odáig, Móricka! Amikor a héj a 2>&1 1>&2 sort olvassa, először a 2>&1 részt hajtja végre. Ezt már ismerjük: a 2 fájlleíró (stderr) ugyanoda kerül, ahová az 1 fájlleíró (stdout). Ezután értelmezi a 2>&1 részt. A stdout (1) ugyanoda kerül, ahová a stderr (2), vagyis a csővezetékbe, mivel a szabályos hibacsatornát már átirányítottuk a szabályos kimenetre.

Ebben az esetben kapóra jönnek a további fájlleírók, 3-tól 9-ig. Ezeket általában nem használjuk. Most azonban az egyiket alkalmazhatjuk „emlékeztetőnek", ami megőrzi, hogy hová „mutatott" a másik fájlleíró. Például a 3>&2 így olvasható: „mutasson a 3 oda, ahová a 2". Miután a 3>&2 segítségével tároltuk a 2 helyét, átirányíthatjuk a 2 fájlleírót. Ezután az 1 oda irányítható, ahová korábban a 2 mutatott (és ahová most a 3 mutat).

A következő parancssorok egyikére lesz szükségünk:

```
$ parancs 3>&2 2>&1 1>&3 | ...
$ var=`parancs 3>&2 2>&1 1>&3`
```

Amikor a folyamat kilép, a rendszer lezárja a megnyitott állományokat. Ennek ellenére mégis az a legbiztosabb, ha lezárjuk a fájlt, amikor már nem használjuk. Így nem jöhet létre ütközés, ha később valami másra használunk egy fájlleírót (például a 3 leírót egy másik program átirányításához, vagy amikor egy alfolyamat próbálja használni a 3 le-

írót). Az `m<&-` segítségével a bemeneti, az `m>&-` segítségével pedig a kimeneti fájlleírókat zárhatjuk le. Ha arra van szükség, a `<&-` lezárja a szabályos bemenetet, a `>&-` pedig a szabályos kimenetet.

## #6 Összetett parancssorok

### Építsünk egyszerű parancsokból egész bekezdéseket összetett (de értelmes) kimutatások készítéséhez

A Linux (és igazából bármilyen Unix) tanulása nagyon hasonlít egy idegen nyelv elsajátításához. A tanulmányok során eljön az a bűvös pillanat, amikor az akadozó, szótagonként kinyögött dadogás elkezdi átadni a helyét az összefüggő és gyakran használt kifejezéseknek. Végül az ember azon kapja magát, hogy egész mondatok és bekezdések buknak ki belőle a Unix nyelvén, miközben az esze maradéktalanul az adott feladat megoldásán jár (és nem egy parancssor írásmódjának (syntax) részletein), ugyanakkor csakúgy, mint a diákok sok időt töltenek azzal, hogy megkérdezzék merre van a mosdó, vagy megértsék tulajdonképpen mi is az a részes eset – a Linux parancssor anyanyelvének elsajátítása is az első félénken kiejtett varázsszavakkal kezdődik.

A parancshéj nagyon elnéző, türelmesen (és ismételten) meghallgatja minden megnyilatkozásunkat, míg végül sikerül eltalálni a helyes formát. Minden program lehet bármely másik program bemenete, ami nagyon érdekes Unix „mondatok" létrehozását teszi lehetővé. Mindig kéznél van a (valószínűleg túl sokszor használt) felfelé nyíl billentyű, amivel apró változtatások során összerakhatunk nagyon bonyolult működést megvalósító parancsokat is.

Tegyük fel, hogy ki kell derítenünk, miért jelez időről időre egy halom hibát a webkiszolgáló. Ha beírjuk a `less error_log` parancsot, látható, hogy sok kisebb hibát okoz egy hiányzó (vagy hibásan hivatkozott) kép:

```
[Tue Aug 27 00:22:38 2002] [error]
➥ [client 17.136.12.171] File does not exist:
➥ /htdocs/images/spacer.gif
```

```
[Tue Aug 27 00:31:14 2002] [error]
➥ [client 95.168.19.34] File does not exist:
➥ /htdocs/image/trans.gif
[Tue Aug 27 00:36:57 2002] [error]
➥ [client 2.188.2.75] File does not exist:
➥ /htdocs/images/linux/arrows-linux-back.gif
[Tue Aug 27 00:40:37 2002] [error]
➥ [client 2.188.2.75] File does not exist:
➥ /htdocs/images/linux/arrows-linux-back.gif
[Tue Aug 27 00:41:43 2002] [error]
➥ [client 6.93.4.85] File does not exist:
➥ /htdocs/images/linux/hub-linux.jpg
[Tue Aug 27 00:41:44 2002] [error]
➥ [client 6.93.4.85] File does not exist:
➥ /htdocs/images/xml/hub-xml.jpg
[Tue Aug 27 00:42:13 2002] [error]
➥ [client 6.93.4.85] File does not exist:
➥ /htdocs/images/linux/hub-linux.jpg
[Tue Aug 27 00:42:13 2002] [error]
➥ [client 6.93.4.85] File does not exist:
➥ /htdocs/images/xml/hub-xml.jpg
```

és így tovább. A naplófigyelő programok (mint az *analog*) jelentik, hogy pontosan hány hiba fordul elő naponta, de ezen kívül kevés részlettel szolgálnak (valószínűleg eleve így figyeltünk föl a hibára). Ha megnézzük közvetlenül a naplóállományt, minden apró részletet feltár, de ez sokkal több adat, semhogy hatékonyan feldolgozhatnánk.

Kezdjük az alapoknál. Van *egyáltalán* olyan hibaüzenet, amit nem hiányzó állomány okoz? Először is tudnunk kellene, hány hibaüzenetet kaptunk ma:

```
$ wc -l error_log
1265 error_log
```

Ebből mennyit okoztak a nem létező fájlok?

```
$ grep "File does not exist:" error_log | wc -l
1265 error_log
```

Kezdetnek nem is rossz. Legalább tudjuk, hogy nincs gond a hozzáférési jogosultságokkal vagy valamelyik tartalmat dinamikusan létrehozó programmal (például a *cgi* parancsfájllal). Ha minden hibát hiányzó állományok okoznak (vagy elírások a HTML-kódban, amelyek hibás fájlnévre hivatkoznak), akkor nem lehet nagyon nagy a baj. Hozzunk létre egy listát az összes fájlnévről, ami a hibás hivatkozásokban szerepel. Nyomjuk meg a felfelé nyilat és a `wc -l` részt töröljük:

```
$ grep "File does not exist:" error_log |
➥ awk '{ print $13} ' | less
```

Éppen erre van szükségünk (csak a 13. mezőre, ami a fájlnév), de várjunk csak egy kicsit! Ugyanaz a néhány állománynév ismétlődik rengetegszer. Na, akár el is küldhetnénk az egészet elektronikus levélben a webfejlesztő csoportnak (mind az 1265 sort), de biztosan nem vennék jó néven a fölösleges levélszemetet. Könnyen elérhető, hogy minden fájl pontosan egyszer szerepeljen:

```
$ grep "File does not exist:" error_log |
➥ awk '{ print $13} ' | sort | uniq | less
```

Ez már sokkal elfogadhatóbb (tegyük a `less` utasítás helyére a `wc -l` parancsot, és meglátjuk hány állomány szerepel a hibaüzenetekben). Lehet, hogy ezek közül egyes fájlokat csupán egyszer kértek a kiszolgálótól, más fájlokat viszont több százszor. Természetesen, ha valahol van egy hibásan írt hivatkozás, sok kérelem érkezhet egyetlen „hiányzó" állományra. A fenti sor azonban semmit sem árul el arról, hogy mely fájlokat kérték le legtöbbször. Ez nem a *bash* dolga; tegyünk próbát egy parancssori `for` ciklussal.

```
$ for x in `grep "File does not exist" error_log |
➥ awk '{ print $13} ' | sort | uniq`; do \
echo -n "$x : "; grep $x error_log | wc -l; done
```

A fordított egyszeres idézőjelekre (`) azért van szükség, hogy végrehajtsuk a parancsot az előző sorból és a kimenetét átadjuk a `for` ciklusnak. A ciklus minden végrehajtásakor az `x` változó értéke az eredeti parancs kimenetének egy újabb sora lesz (vagyis a következő hi-

ányzó fájl neve). Ezután a `grep` megkeresi a fájlnevet az *error_log* állományban, majd megszámoljuk hányszor fordul elő. Végül az `echo` mindezt viszonylag csinos formában kiírja.

Azért csak viszonylag csinos, mert a kimenet tele van egyszeri találatokkal, nagyon egyenetlen és még csak sorrendbe se állítottuk. Rendezzük sorba a találatok száma szerint, kezdve a legtöbbször előforduló fájlnevekkel, és csak a húsz leggyakrabban kért hiányzó fájl nevét mutassuk meg:

```
$ for x in `grep "File does not exist" error_log |
➥ awk '{ print $13} ' | sort | uniq`; do \
grep $x error_log | wc -l | tr -d '\ n';
➥ echo " : $x"; done | sort +2 -rn | head -20
```

Így már *sokkal* jobb, és nem is kerül lényegesen több gépelésbe.
A `tr` parancs a *wc* program kimenetéből tünteti el az újsor karaktereket (hogy miért nincs a parancsnak erre egy kapcsolója, soha nem fogom megérteni). A kimenet valahogy így fest:

```
595 : /htdocs/images/pixel-onlamp.gif.gif
156 : /htdocs/image/trans.gif
139 : /htdocs/images/linux/arrows-linux-back.gif
68 : /htdocs/pub/a/onjava/javacook/images/spacer.gif
50 : /htdocs/javascript/2001/03/23/examples/target.gif
```

Ebből a kivonatból jól látható, hogy a hibák csaknem fele egy népszerű weblapon található elírás következménye (figyeljük meg a kettőzött *.gif.gif* végződést az első sorban). Valószínűleg a második is elírás (*images/*-nek kellene lennie az *image/* helyett). A többit találja ki a webes csoport:

```
$ ( echo "Kimutatás a 20 leggyakrabban kért
➥ 'hiányzó' fájlról az error_log-ból. "; echo; \
for x in `grep "File does not exist" error_log |
➥ awk '{ print $13} ' | sort | uniq`; do \
grep $x error_log | wc -l | tr -d '\ n';
➥ echo " : $x"; done | sort +2 -rn | head -20 )\
| mail -s "Kimutatás a hiányzó fájlokról"
➥ webmaster@oreillynet.com
```

Esetleg szükség lehet egy nyomtatott változatra a heti fejlesztői megbeszélésre:

```
$ for x in `grep "File does not exist" error_log |
➥ awk '{ print $13} ' | sort | uniq`; do \
grep $x error_log | wc -l | tr -d '\ n';
➥ echo " : $x"; done | sort +2 -rn | head -20 \
| enscript
```

### Javított kiadás

Aki hozzászokik, hogy parancsok sorát fűzze össze, a végtelenségig folytathatja a kimenetek átirányítását, és tetszőleges kivonatot állíthat elő élő adatfolyamokból. Amikor azon kapjuk magunkat, hogy egy feladatot újra meg újra végrehajtunk, természetesen eszünkbe jut, hogy a parancsokat egy héjprogramba foglaljuk (esetleg a hatékonyság kedvéért újraírjuk Perl vagy Python nyelven, mivel minden egyes | azt jelenti, hogy újabb programot futtatunk). Egy korszerű (nem túlterhelt) gépen aligha vesszük észre a különbséget, de ezt jó szokásnak tartják, mivel ezzel az eljárással letisztázhatjuk a megoldásokat, amiket összebarkácsoltunk. S mint az közismert, a parancssor mindig tág teret enged a barkácsolásnak.

## #7 Cseles fájlnevek kezelése az xargs segítségével

### Bánjunk el egyszerre a több szóközt vagy más furcsa karaktert tartalmazó fájlnevekkel!

Ha sok szóközt, zárójelet és más „tiltott" karaktert tartalmazó állománynévvel kell dolgoznunk, fárasztó feladat vár ránk. Ez a gond úgy tűnik, gyakran merül fel manapság, a digitális zenetárolás robbanásszerű terjedése idején. Szerencsére a *bash* fájlnévkiegészítő szolgáltatásával egyszerűen kezelhető egyszerre egy állomány:

```
rob@catlin:~/Music$ ls
Crog - The Lone Deranger
Misc - Pure Disco
rob@catlin:~/Music$ rm -rf Misc[TAB]
rob@catlin:~/Music$ rm -rf Misc\  -\  Pure\  Disco/
```

A TAB lenyomása a fenti példában a *[TAB]* helyén a parancssort kiegészíti az alatta látható sorra, szabályosan jelölve a fájlnévben szereplő

különleges karaktereket. Ez rendben is van, ha egyszerre csak egy fájlnévvel akad dolgunk, de mi a helyzet akkor, ha tömegesen akarjuk feldolgozni az állományokat (például egy halom MP3 nevébe kellene beszúrnunk a lemez nevét)? Vegyünk egy példát:

```
rob@catlin:~/Music$ cd Hall[TAB]
rob@catlin:~/Music$
➥ cd Drog\ -\ The\ Lone\ Deranger/
rob@catlin:~/Music/Drog - The Lone Deranger$ ls
Drog - 01 - Dementia.mp3
Drog - 02 - Snakey Shaker.mp3
Drog - 03 - Trancespotter.mp3
Drog - 04 - Horrorgram.mp3
Drog - 05 - Snarling (Remix).mp3
Drog - 06 - Gamma Goblins Pt. 2.mp3
Drog - 07 - Deranger.mp3
Drog - 08 - Jiggle of the Sphinx.mp3
rob@catlin:~/Music/Drog - The Lone Deranger$
```

Ha sok fájlnévvel szeretnénk dolgozni egyszerre, ügyeskednünk kell. Számos általánosan használt segédprogram elválasztásnak veszi a fájlnevekben lévő szóközöket (több mezőre bontja a bemenetet, mint kellene), és teljesen lerobban, ha egy ) (zárójel) vagy egy { (kapcsos zárójel) akad az útjába. Olyan elválasztó karakterre van szükség, ami biztosan nem fordul elő fájlnevekben.

Szerencsére az *xargs* program képes a NULL karaktert elválasztóként értelmezni, ha szépen megkérjük rá. Figyeljük meg a következő parancsállományt:

### Lista: albumize

```
#!/bin/sh

if [ -z "$ALBUM" ]; then
echo 'Meg kell adni az ALBUM nevet
      ➥ (pl. export ALBUM="Greatest Hits")'
exit 1
fi

for x in *; do
echo -n $x; echo -ne '\ 000'
```

```
echo -n `echo $x|cut -f 1 -d '-'`
echo -n " - $ALBUM - "
echo -n `echo $x|cut -f 2- -d '-'`; echo -ne '\
000'
done | xargs -0 -n2 mv
```

Tulajdonképpen két cselt alkalmazunk: először összeállítunk egy listát, amelyben a könyvtárban lévő összes fájlnevet a NULL karakterrel elválasztva követi az a név, amire át fogjuk nevezni. Ezután az egész listát két kapcsolóval adjuk át az *xargs* programnak: a -0 azt jelenti, hogy (újsor vagy szóköz helyett) a NULL karakter az elválasztó, a -n2 pedig arra utasítja, hogy egy körben két kapcsolót adjon át a programnak (mv).

A *~/bin/albumize* néven mentsük a parancsállományt. Mielőtt futtatnánk, állítsuk be az ALBUM változót arra a névre, amit be szeretnénk szúrni a fájlnevekbe az első - (mínuszjel) után. Tegyünk egy próbát!

```
rob@catlin:~/Music/Drog - The Lone Deranger$
➥ export ALBUM="The Lone Deranger"
rob@catlin:~/Music/Drog - The Lone Deranger$
➥ albumize
rob@catlin:~/Music/Drog - The Lone Deranger$ ls
Drog - The Lone Deranger - 01 - Demention.mp3
Drog - The Lone Deranger - 02 - Snakey Shaker.mp3
Drog - The Lone Deranger - 03 - Trancespotter.mp3
Drog - The Lone Deranger - 04 - Horrorgram.mp3
Drog - The Lone Deranger - 05 - Snarling (Remix).mp3
Drog - The Lone Deranger - 06 - Gamma Goblins Pt.
                                ➥ 2.mp3
Drog - The Lone Deranger - 07 - Deranger.mp3
Drog - The Lone Deranger - 08 - Jiggle of the
                                ➥ Sphinx.mp3
rob@catlin:~/Music/Drog - The Lone Deranger$
```

Mit csináljunk, ha el szeretnénk távolítani az albumnevet? Használjuk a következő héjprogramot, és adjuk neki a *~/bin/dealbumize* nevet:

```
#!/bin/sh

for x in *; do
echo -n $x; echo -ne '\ 000'
```

```
echo -n `echo $x|cut -f 1 -d '-'`; echo -n ' - '
echo -n `echo $x|cut -f 3- -d '-'`; echo -ne '\
000'
done | xargs -0 -n2 mv
```

Egyszerűen csak futtassuk (nem kell az ALBUM változót megadni):

```
rob@catlin:~/Music/Drog - The Lone Deranger$
➥ dealbumize
rob@catlin:~/Music/Drog - The Lone Deranger$ ls
Drog - 01 - Demention.mp3
Drog - 02 - Snakey Shaker.mp3
Drog - 03 - Trancespotter.mp3
Drog - 04 - Horrorgram.mp3
Drog - 05 - Snarling (Remix).mp3
Drog - 06 - Gamma Goblins Pt. 2.mp3
Drog - 07 - Deranger.mp3
Drog - 08 - Jiggle of the Sphinx.mp3
rob@catlin:~/Music/Drog - The Lone Deranger$
```

A -0 kapcsoló másik népszerű felhasználási módja, amikor a *find* program -print0 kapcsolójával együtt használjuk (a -print0 az újsor helyett természetesen a találatokat írja ki NULL karakterekkel elválasztva). Ha egy csővezetékre kötjük a *find* és az *xargs* programot, bármit megtehetünk bármennyi állománnyal, és nem kell tartanunk a hírhedt Argument list too long hibaüzenettől sem:

```
rob@catlin:~/Pit of too many files$ ls
bash: /bin/ls: Argument' list too long
```

A *find–xargs* páros gyorsan elbánik ezekkel a fájlokkal, bármi is legyen a nevükben:

```
rob@catlin:/Pit of too many files$
➥ find -type f -print0 | xargs -0 ls
```

Ha törölni akarjuk az állományokat, egyszerűen cseréljük az ls parancsot rm-re és annyi nekik.

# #8 Megváltoztathatatlan állományok az ext2, illetve az ext3 fájlrendszerben

## A fájlok, amelyeket még a rendszergazda sem bánthat

Egy találós kérdés következik: tegyük föl, hogy a */tmp* takarítása közben gondunk akad.

```
root@catlin:/tmp# rm -rf junk/
rm: cannot unlink `junk/stubborn.txt':
➥ Operation not permitted
rm: cannot remove directory `junk':
➥ Directory not empty
root@catlin:/tmp# cd junk/
root@catlin:/tmp/junk# ls -al
total 40
drwxr-xr-x 2 root root 4096 Sep 4 14:45 ./
drwxrwxrwt 13 root root 4096 Sep 4 14:45 ../
-rw-r--r-- 1 root root 29798 Sep 4 14:43 stubborn.txt
root@catlin:/tmp/junk# rm ./stubborn.txt
rm: remove write-protected file `./stubborn.txt'? y
rm: cannot unlink `./stubborn.txt':
➥ Operation not permitted
```

Mi történt? Most akkor én vagyok a rendszergazda, vagy nem én vagyok? Próbáljuk meg a fájl helyett a tartalmát törölni:

```
root@catlin:/tmp/junk# cp /dev/null stubborn.txt
cp: cannot create regular file `stubborn.txt':
➥ Permission denied
root@catlin:/tmp/junk# > stubborn.txt
bash: stubborn.txt: Permission denied
```

Nos, a */tmp* biztosan nincs csak olvasható módban befűzve. Vajon mi folyik itt?

Az `ext2` és `ext3` fájlrendszerben jó néhány fájljellemző beállítható a `chmod` paranccsal elérhető szabályos biteken túl. Ha még nem ismernénk ezeket, nézzük meg a `chattr` és társa, az `lsattr` súgóoldalait (man pages).

A beállítások közül az egyik különösen hasznos, ez a `-i` (`immutable`) kapcsoló. Ha ez a bit be van állítva, letilt minden kísérletet az állomány törlésére, átnevezésére, felülírására és a hozzátoldásra. A rendszergazdai jogosultság sem ér semmit, ha a `-i` kapcsoló aktív:

```
root@catlin:/tmp/junk# ln stubborn.txt another.txt
ln: creating hard link `another.txt'
➥ to `stubborn.txt': Operation not permitted
```

Az állományra érvényes kiegészítő beállításokat az `lsattr` paranccsal nézhetjük meg:

```
root@catlin:/tmp/junk# ln stubborn.txt another.txt
ln: creating hard link `another.txt'
➥ to `stubborn.txt': Operation not permitted
```

A bitek beállítását pedig a `chattr` paranccsal végezhetjük, a' la `chmod`:

```
root@catlin:/tmp/junk# chattr -i stubborn.txt
root@catlin:/tmp/junk# rm stubborn.txt
root@catlin:/tmp/junk#
```

Ez hihetetlenül hasznos lehet, mivel újabb biztonsági réteget adhatunk olyan fájloknak, amiket soha nem akarunk megváltoztatni (mondjuk a */etc/rc.d/** vagy számos beállításfájl esetében). Bár nem sok esélyünk van, ha egy behatoló rendszergazdai jogosultságokat szerez a gépen, a megváltoztathatatlan állományok jó eséllyel védettek az egyszerű, más folyamatokon keresztül kivitelezett felülírásos támadásokkal szemben, még akkor is, ha a folyamat rendszergazdai jogosultsággal fut.

Beállítható tömörítés, biztonságos törlés, törölhetetlenség, szinkron írás és még néhány hasznos tulajdonság. Amikor ezeket leírom, a kiegészítő beállítások közül még sokat nem valósítottak meg, de érdemes figyelemmel kísérni az `ext` állományrendszer fejlesztéseit.

# #9 A fordítások gyorsítása

## Adjunk munkát minden processzornak párhuzamos fordítással

Ha többprocesszoros (SMP) gépet használunk megfelelő mennyiségű RAM-mal, általában jelentős előnnyel jár, ha a programok fordítását párhuzamos feldolgozással végezzük. A `make` soros végrehajtásához képest (ez az alapbeállítás) a párhuzamos feldolgozás óriási teljesítménynövekedést jelent.

A `make` parancs `-j` kapcsolója szolgál egynél több fordítási folyamat engedélyezésére:

```
rob@mouse:~/linux$ make -j4; make -j4 modules
```

Nem minden programrendszert terveztek úgy, hogy elviseljék a párhuzamos fordítást; ezeknél gondot okozhat, ha egyes részek előbb készülnek el, mint a felmenő függőségeik fordítása. Ha fordítási hibákba botlunk, a legbiztosabb, ha mindent elölről kezdünk a `-j` kapcsoló nélkül.

Az összehasonlítás kedvéért álljon itt néhány mérési eredmény, amely egy más feladattal nem terhelt, két Pentium III 600 MHz-es processzorral és 1 GB memóriával rendelkező gépen készült. Minden alkalommal *bzImage* készült a Linux 2.4.19 rendszermag forrásából (a `STDOUT`-ot átirányítva a */dev/null* eszközre) és a következő mérés előtt mindig eltávolítottam a forrásfát.

**`time make bzImage:`**

```
real 7m1.640s
user 6m44.710s
sys 0m25.260s
```

**`time make -j2 bzImage:`**

```
real 3m43.126s
user 6m48.080s
sys 0m26.420s
```

```
time make -j4 bzImage:

real 3m37.687s
user 6m44.980s
sys 0m26.350s

time make -j10 bzImage:

real 3m46.060s
user 6m53.970s
sys 0m27.240s
```

Amint látható, jelentős teljesítménynövekedés érhető el pusztán a `-j2` kapcsoló hozzáadásával. 7 percről 3 perc 43 másodpercre mérséklődött a tulajdonképpeni fordítási idő. A `-j4` megint megtakarított öt másodpercet, de a `-j10` néhány másodpercet rontott a teljesítményen. Figyeljük meg, hogy a felhasználói és rendszeridő szinte másodpercre egyezik az összes esetben. Végeredményben mindig ugyanakkora bithalmot kell átlapátolni, de a `-j` kapcsoló többprocesszoros gépen lehetővé teszi, hogy több kubikos férjen oda a halomhoz.

Legvégül minden bit az örök bitvadászmezőkre jut. De fel a fejjel, ha másra nem is, egy kis fűtésnek mindig jók lesznek a teljesítménypróbák.

## #10 Otthonos parancssori környezet

### Tegyük kényelmesebbé a bash használatát környezeti változókkal

A *bash* súgójának olvasgatása elég kimerítő elfoglaltság, főként, ha nem tudjuk pontosan mit is keresünk. De ha sikerül időt szakítanunk rá, a *bash* súgóoldala hasznos olvasmánynak bizonyul. Ez a héj hemzseg a titokzatos (és csodálatosan hasznos) lehetőségektől, amelyek legtöbbje alapértelmezetten egyszerűen ki van kapcsolva.

Kezdjük néhány előnyös környezeti változóval és pár hasznos értékkel, amire beállíthatjuk őket:

```
export PS1=`echo -ne "\ 033[0;34m\ u@\ h:\
033[0;36m\ w\ 033[0;34m\ $\ 033[0;37m "`
```

Amint az ismeretes, a PS1 változó az alapértelmezett parancssorjelet (prompt) állítja be, és értelmezi a vezérlőkaraktereket, mint a *\u* (ami a felhasználói név) és a *\w* (jelenlegi munkakönyvtár). Amint az kevéssé köztudomású a héj parancssorjelébe ANSI vezérlőkarakterek is kerülhetnek, amelyekkel színt vihetünk a parancssorjel megjelenésébe. Az egészet fordított egyszeres idézőjelek (`) közé zárjuk, hogy az echo előállíthassa a bűvös ASCII vezérlőkaraktert. A parancs egyszer fut le és az eredményt a PS1 változó tárolja. Nézzük meg még egyszer ugyanezt a parancssort, kiemelve mindent, ami nem ANSI-kód:

```
export PS1=`echo -ne "\ 033[0;34m\ u@\ h:\
033[0;36m\ w\ 033[0;34m\ $\ 033[0;37m "`
```

Fölismerhetjük a mindenki által szeretett és megszokott \u@\h:\w\$ parancssorjelet. A pontosvesszők utáni számok változtatásával bárki kedve szerint állíthatja be az egyes részek színét.

Hasonló szellemben következzen egy hasznos parancs, ami közvetlenül az előtt kerül végrehajtásra, hogy a *bash* parancssort megkapjuk:

```
export PROMPT_COMMAND='echo -ne "\ 033]0;${ USER}
@${ HOSTNAME} : ${ PWD} \ 007"'
```

(Most nincs szükség fordított idézőjelekre, mert a bash a változóból egy parancsot vár, nem pedig egy karaktersorozatot.) Ebben az esetben a vezérlőkarakterek sora az a bűvös karakterlánc, ami a címsort beállítja a legtöbb terminálablakon (például az xterm, rxvt, eterm, gnometerm stb.). Minden, ami a pontosvessző és a \007 között található, megjelenik a címsorban, ahányszor csak új parancssort kapunk. A példában megjelenik a felhasználói név, a gép neve, amelyre a felhasználó bejelentkezett és a pillanatnyilag érvényes munkakönyvtár. Ez igencsak jól jöhet, mivel egyetlen pillantással megállapítható (akár a vim használata közben is), hogy épp melyik gépre jelentkeztünk be, és melyik könyvtárba készülünk menteni az állományt. Ha valós időben akarjuk frissíteni a címsort, nem csupán minden új parancssor megjelenésével, nézzük meg [#59] „Hogyan végeztessünk több munkát a címsorral, hogy ne kerüljön nagy erőfeszítésbe" trükköt.

Előfordult-e vajon mással is, hogy túl sokszor nyomta le egymás után a ^D billentyűkombinációt, és azon vette észre magát, hogy kilépett a rendszerből? Beállíthatjuk, hogy a bash tetszőleges számú egymás utáni ^D leütést figyelmen kívül hagyjon:

```
export IGNOREEOF=2
```

Ennek hatására a *bash* a Snark-szabály szerint jár el („Amit háromszor mondok, az igaz") és csak három egymás utáni ^D hatására léptet ki. Ha ez valakinek túl kevés lenne, állítson be nyugodtan 101 leütést, és a *bash* engedelmesen számon tartja a leütéseket.

Nagyon hasznos lehet, ha a saját könyvtárunk egyik alkönyvtára rajta van a keresési útvonalon (tarthatunk itt parancsállományokat, közvetett hivatkozásokat, illetve tetszőleges programokat). Erre a célra hagyományosan a saját könyvtárban elhelyezett *bin* alkönyvtárat használjuk. A *bash* egyik szolgáltatása a ~ helyettesítés:

```
export PATH=$PATH:~/bin
```

Használatával az útvonal megadása mindig helyes lesz, még akkor is, ha a saját könyvtárat áthelyeznénk (és akkor is, ha ugyanezt a sort több gépen, esetleg más-más saját könyvtárral használjuk – mint a *movein.sh* esetében. Lásd még a [#72] „Intézzük el a dolgokat gyorsan a *movein.sh* segítségével").

Jó tudni, hogy éppen úgy, ahogyan a programok elérési útját a `PATH` változó (a súgóoldalak elérési útját a `MANPATH` változó) tárolja, a könyvtárak keresési útvonalát a `CDPATH` változó adja meg minden alkalommal, amikor a `cd` parancsot kiadjuk. Az alapértelmezés a . (pont), de bármi megadható:

```
export CDPATH=.:~
```

Ennek hatására a `cd` nemcsak a pillanatnyi munkakönyvtárban, hanem a saját könyvtárban is keresni fogja azt az alkönyvtárat, amelyikre váltani akarunk. Például:

```
rob@caligula:~$ ls
bin/ devel/ incoming/ mail/ test/ stuff.txt
rob@caligula:~$ cd /usr/local/bin
rob@caligula:/usr/local/bin$ cd mail
bash: cd: mail: No such file or directory
rob@caligula:/usr/local/bin$ export CDPATH=.:~
rob@caligula:/usr/local/bin$ cd mail
/home/rob/mail
rob@caligula:~/mail$
```

Korlátlan számban adhatunk meg a CDPATH változóban könyvtárakat, amelyekben keresni akarunk : (kettősponttal) elválasztva (mint a PATH és MANPATH változókban).

Mindenki ismeri a felfelé nyíl és a `history` parancs működését. Mi a helyzet akkor, ha valamilyen kényes adatot írtunk be a parancssorba? Tegyük fel, hogy elcsúszunk a gépeléssel, és véletlenül beírunk egy jelszót oda, ahová egy parancsot szántunk. Ezt a kis balesetet kilépéskor hűen rögzíti a *~/.bash_history* fájl, itt azonban egy gátlástalanabb felhasználó esetleg rábukkanhat. A *.bash_history* átírása egy szövegszerkesztővel nem oldja meg a gondot, mivel a fájl minden kilépéskor felülíródik. A megőrzött parancsok gyors törlésére használható a következő parancs:

```
export HISTSIZE=0
```

Ez törli a *bash* emlékezetét és kilépéskor egy üres *.bash_history* állományt hoz létre. A következő belépéskor a parancssor emlékezete tiszta lappal indul újra, a működésében azonban semmi nem változik. Máskor vigyázzunk jobban!

Akadt már dolgunk olyan felhasználókkal, akik belépnek egy gépre, azután leválasztják hordozható gépüket és hazamennek anélkül, hogy kilépnének? Ha rendszeresen látunk több napja bejelentkezett felhasználókat a `w` parancs kiadása után, a következő változót állítsuk be munkakörnyezetükben:

```
export TMOUT=600
```

A TMOUT változó azt adja meg, hogy a *bash* meddig várjon bevitelre a parancssorban, mielőtt önműködően kilépteti a felhasználót. Ez nem segít, ha a felhasználók egy `vi` ablakban ücsörögnek, de kiszűri a tétlenül várakozó parancssort hátrahagyó felhasználókat. Tíz perc egy kicsit kevés lehet egyes felhasználóknak, de őket udvariasan emlékeztethetjük rá, hogy beállíthatnak más értéket is, ha az alapértelmezés nem felel meg nekik.

Ezzel fölvetődik egy fontos kérdés: pontosan hogyan is tehetjük tartóssá a munkakörnyezet beállításait? A *bash* indulásakor több fájlt is megnéz, attól függően, hogy a héj belépéskor indul vagy egy másik héjból hívtuk.

A *bash(1)* ezt írja a belépési héjról:

„...először beolvassa és végrehajtja a */etc/profile* fájlban található parancsokat, ha létezik ez a fájl. Ez után a *~/.bash_profile*, *~/bash_login* és a *~/.profile* fájlokat keresi, ebben a sorrendben, beolvassa és végrehajtja a parancsokat az első fájlból, amelyik létezik és olvasható... Amikor egy belépési héj kilép, a bash beolvassa és végrehajtja a parancsokat a *~/.bash_logout* fájlból, ha létezik ilyen."

Minden más héj esetében: a bash beolvassa és végrehajtja a parancsokat a *~/.bashrc* fájlból, ha az létezik.

Annak pontos meghatározása, hogy mi is az a belépési héj (és egy egész halom tudnivaló arról a környezetről, amiben nap mint nap dolgozunk) a *bash(1)* súgóoldalon található.

## #11 A setuid és setgid programok kiküszöbölése

A linuxos kiszolgálók üzemeltetése során jó szolgálatot tett nekem az az alapelv, hogy folyamatosan föl kell tenni a kérdést: „Mit is akarok megvalósítani? Van-e értelme nyomtatási szolgáltatást telepíteni egy webkiszolgálóra? Kell-e telepíteni a *gpm*-et egy konzol nélküli gépre?"

Általában a felesleges programcsomagok csak szükségtelenül foglalják a helyet a merevlemezen, de a *setuid* és *setgid* programokkal sokkal rosszabbul is járhatunk.

Noha a Linux-terjesztések összeállítói keményen dolgoznak annak érdekében, hogy a *setuid* és *setgid* programok minden ismert biztonsági rését kiküszöböljék, nagyjából egy-két havonta csak fölbukkan néhány új és váratlan betörési lehetőség. Különösen, ha a kiszolgálóhoz nem csak egy felhasználónak van parancssori hozzáférése, rendszeresen felül kell vizsgálni a telepített *setuid* és *setgid* programokat. Jó eséllyel meglepetést fog okozni, hogy milyen sok található ezekből a gépen.

A következő parancs egyike azoknak, amellyel az összes *setuid* vagy *setgid* bittel jelzett programot megtalálhatjuk:

```
root@catlin:~# find / -perm +6000 -type f -exec
➥ ls -ld { }  \ ; > setuid.txt &
```

Ez a parancs létrehoz egy *setuid.txt* nevű állományt, amiben megtalálható az összes, a keresési feltételnek megfelelő fájl minden részlete. Tanácsos végigböngészni a listát és eltávolítani az s bitet minden olyan segédprogramról, amit nem használunk.

Nézzük végig, mit találhatunk egy átlagos gépen:

```
-rws--x--x 1 root bin 35248 May 30 2001
➥ /usr/bin/at
-rws--x--x 1 root bin 10592 May 30 2001
➥ /usr/bin/crontab
```

Ez nem meglepő. Az `at` és a `cron` démonnak rendszergazdai jogosultságokra van szüksége hozzá, hogy annak a felhasználónak a nevében hajtsa végre az `at` vagy `cron` feladatokat, aki a feladatok végrehajtását kérte. Aki üldözési mániában szenved, és nem használja ezeket a szolgáltatásokat, a *setuid* biteket a következő paranccsal távolíthatja el:

```
# chmod a-s /usr/bin/{ at,crontab}
```

Általában véve elég rossz ötlet letiltani a `cron` démont (mivel számos rendszer az időzített végrehajtástól függ). De mikor használtuk utoljára az `at` démont? Tudják egyáltalán a felhasználóink, hogy mire való? Én személy szerint jól használható egyszerűsítésnek tartom az `at` használatát teljes értékű `cron` feladatok beállítása helyett, és nem szívesen válnék meg tőle. De ha egy gépen nincs rá kifejezetten szükség, nem árt kihúzni a méregfogát. A *setuid* bit eltávolítása után a parancsokat a rendszergazda még mindig használhatja, csak az egyszerű felhasználók nem érhetik el.

```
-rws--x--x 1 root bin 11244 Apr 15 2001
➥ /usr/bin/disable-paste
```

Ez a *gpm* (a Linux-konzolon használható egérmeghajtó) csomag része. Van-e egér csatlakoztatva a gép konzoljához? Használjuk-e szöveges módban? Ha nem, miért hagynánk meg egy *setuid* programot, amit soha sem indítunk el?

```
-r-s--s--x 1 root lp 14632 Jun 18 2001
➥ /usr/bin/lpq
-r-s--s--x 1 root lp 15788 Jun 18 2001
➥ /usr/bin/lpr
-r-s--s--x 1 root lp 15456 Jun 18 2001
➥ /usr/bin/lprm
-r-xr-s--x 1 root lp 23772 Jun 18 2001
➥ /usr/sbin/lpc
```

Ezek mind a nyomtatási alrendszer részét képezik. Nyomtat-e egyáltalán ez a gép az `lp`-vel?

```
-rws--x--x 1 root bin 33760 Jun 18 2000
➥ /usr/bin/chage
-rws--x--x 1 root bin 29572 Jun 18 2000
➥ /usr/bin/chfn
-rws--x--x 1 root bin 27188 Jun 18 2000
➥ /usr/bin/chsh
-rws--x--x 1 root bin 35620 Jun 18 2000
➥ /usr/bin/passwd
```

Ezekre szükségük van a felhasználóknak, hogy beállíthassák a jelszavukat, a héjat és a `finger` adatokat. Használ-e a gép egyáltalán `finger` adatot? Ha a legtöbb géphez hasonlóan létezik egy névsor valahol máshol (valószínűleg a weben), ami nincs összehangolva a felhasználók GECOS mezőjével, akkor az adatok valószínűleg használhatatlanok (kivéve a felhasználók „valódi" nevét, amit még mindig használ egy pár levelezőprogram). Tényleg meg kell engednünk, hogy a felhasználók a rendszergazda beavatkozása nélkül a saját szakállukra változtassák ezeket az adatokat?

```
-r-xr-sr-x 1 root tty 9768 Jun 21 2001
➥ /usr/bin/wall
-r-xr-sr-x 1 root tty 8504 Jun 21 2001
➥ /usr/bin/write
```

A `wall`-nak és a `write`-nak is a `tty` csoport nevében kell futnia, mivel más felhasználók termináljára fogunk írni. Ez általában biztonságos művelet, de visszaélhetnek vele azok a hitszegők, akik örömüket lelik abban, hogy csúnya dolgokat (vagy *sok* csúnya dolgot) írjanak mások termináljára. Ha ezt a szolgáltatást nem akarjuk mindenki számára hozzáférhetővé tenni, miért ne tiltsuk le a *setgid* bitet? Ha ezt tesszük, a rendszergazda még mindig küldhet `wall` vagy `write` üzeneteket (például ilyen üzenet az, amit a `shutdown` program küld újraindításkor).

```
-rwsr-xr-x 1 root bin 14204 Jun 3 2001
➥ /usr/bin/rcp
-rwsr-xr-x 1 root bin 10524 Jun 3 2001
➥ /usr/bin/rlogin
-r-sr-xr-x 1 root bin 7956 Jun 3 2001 /usr/bin/rsh
```

Az `r*` parancsok az `ssh` előtti (talán nem is olyan régi) időkből maradtak. Szükséges-e `r` parancsokat biztosítanunk a felhasználóink számára? Van-e valami, amit az `ssh` és az `scp` nem tud, és amihez feltétlenül szükséges az `rsh` és az `rcp`? Több mint valószínű, hogy aki egyszer elkezdi az `ssh`-t használni, egy idő után gondolni sem fog az `r` parancsokra, és nyugodtan eltávolíthatja a használaton kívüli *setuid rsh* csomagot, ami bármikor időzített bombának bizonyulhat.

Ha az SSH-t érdekes dolgokra szeretnénk használni, a könyvben több helyen is találhatók vele foglalkozó trükkök.

```
-r-sr-xr-x 1 root bin 10200 Jun 3 2001
➥ /usr/bin/traceroute
-r-sr-xr-x 1 root bin 15004 Jun 3 2001 /bin/ping
```

A *traceroute* és a *ping* programnak szüksége van a *setuid* rendszergazdai bitre, hogy ICMP-csomagokat állíthasson elő. Ha azt akarjuk, hogy a felhasználóink is futtathassák ezeket a hálózati ellenőrző eszközöket, akkor meg kell hagynunk *setuid* programokat a rendszergazdának. Egyébként pedig távolítsuk el a *setuid* bitet, és csak a rendszergazda futtathatja majd a *traceroute* és a *ping* programot.

```
-r-sr-sr-- 1 uucp uucp 83344 Feb 10 2001
➥ /usr/bin/uucp
-r-sr-sr-- 1 uucp uucp 36172 Feb 10 2001
➥ /usr/bin/uuname
-r-sr-sr-- 1 uucp uucp 93532 Feb 10 2001
➥ /usr/bin/uustat
-r-sr-sr-- 1 uucp uucp 85348 Feb 10 2001
➥ /usr/bin/uux
-r-sr-sr-- 1 uucp uucp 65492 Feb 10 2001
➥ /usr/lib/uucp/uuchk
-r-sr-sr-- 1 uucp uucp 213832 Feb 10 2001
➥ /usr/lib/uucp/uucico
-r-sr-sr-- 1 uucp uucp 70748 Feb 10 2001
➥ /usr/lib/uucp/uuconv
-r-sr-sr-- 1 uucp uucp 315 Nov 22 1995
➥ /usr/lib/uucp/uusched
-r-sr-sr-- 1 uucp uucp 95420 Feb 10 2001
➥ /usr/lib/uucp/uuxqt
```

Mikor fordult elő utoljára, hogy UUCP-protokollal kapcsoltunk össze gépeket? Állítottunk már be valaha UUCP rendszert? Tíz éve vagyok hálózati rendszergazda, és ez idő alatt soha nem találkoztam élő UUCP telepítéssel. Ez nem azt jelenti, hogy az UUCP nem hasznos, csak azt, hogy manapság az állandóan kapcsolt TCP/IP-hálózatok

idején az UUCP nagyon ritka jelenséggé vált. Ha nem használjuk az UUCP-t akkor nem túl sok értelme van az ezt támogató *setuid* és *setgid* programokat a gépen hagyni.

Lehetséges-e a fenti programok bármelyikét rendszergazdai jogosultságok megszerzésére (vagy más, magasabb szintű jogok megszerzésén alapuló támadásra) használni? Fogalmam sincs. De azt tudom, hogy a felesleges *setuid* és *setgid* jogosultságok eltávolításával csökkenthető annak az esélye, hogy a gépet feltörjék, ha mégis felfedeznek egy biztonsági rést.

Ezt a tippet próbáltam folyamatként és nem trükként bemutatni. Ez a felsorolás semmi esetre sem teljes. A kiszolgálók üzembe helyezésénél mindig tartsuk szem előtt, hogy mi a feladat, és ennek megfelelően építsük fel a rendszert. Amikor csak lehetséges, távolítsuk el azokat az engedélyeket (vagy akár egész programcsomagokat), amelyek felesleges szolgáltatásokat nyújtanak. Mindig nézzük meg a súgóoldalt, ha nem tudjuk, hogy egy programnak mi a feladata, és miért van *setuid* beállítással telepítve (és ha a súgótól nem lettünk okosabbak, még mindig ott van a forráskód).

## #12 Dolgozzon a sudo!

### A sudo segítségével másokra bízhatjuk a piszkos munkát, de nem kell átadnunk a jelszót

A sudo program segítségével másokra bízhatjuk a rendszerfelügyelet egy részét anélkül, hogy teljes rendszergazdai hozzáférést adnánk. Ez egy *setuid* rendszergazdai program, amely parancsokat hajt végre az erre felhatalmazott felhasználó nevében, miután megadta a jelszavát.

Rendszergazdaként a */usr/sbin/visudo* használatával szerkeszthetjük a sudo futtatására felhatalmazott felhasználók listáját. Az alapértelmezett lista valahogy így néz ki:

```
root  ALL=(ALL)  ALL
```

Sajnos sok rendszergazda ezt a bejegyzést mintának tekinti és minden rendszerkarbantartónak teljes rendszergazdai jogosultságot ad:

```
root  ALL=(ALL)  ALL
rob  ALL=(ALL)  ALL
jim  ALL=(ALL)  ALL
david  ALL=(ALL)  ALL
```

Bár ez lehetővé teszi, hogy rendszergazdai hozzáférést adjunk a rendszergazdai jelszó terjesztése nélkül, igazából csak akkor használható módszer, ha minden sudo felhasználó teljes mértékben megbízható. Megfelelően beállítva a sudo program lehetővé teszi tetszőleges parancs futtatását tetszőleges felhasználói azonosítóval.

A sudo sorok formája a következő:

**felhasznalo gép**=(*felhasználónév*) **parancs**

Az első oszlop a sudo felhasználó azonosítója. A második azokat a gépeket adja meg, amelyeken érvényes ez a sudo bejegyzés. Ez lehetővé teszi, hogy egy sudo beállítást egyszerűen használjunk több gépen.

Tegyük föl, hogy egy fejlesztőnek rendszergazdai jogosultságokra van szüksége a fejlesztői gépen, de más kiszolgálókon nem:

```
peter  beta.oreillynet.com=(ALL)  ALL
```

A következő oszlop (zárójelben) azt a felhasználót adja meg, akinek a nevében a sudo parancsokat hajthat végre. Ez jól jön, ha egyes felhasználóknak lehetővé akarjuk tenni, hogy más, nem rendszergazdai felhasználóként futtassanak programokat:

```
peter  lists.oreillynet.com=(mailman)  ALL
```

Végül az utolsó oszlopban adhatók meg a parancsok, amiket az adott felhasználó kiadhat:

```
david  ns.oreillynet.com=(bind)
/usr/sbin/rndc,/usr/sbin/named
```

Ha azon kapjuk magunkat, hogy hosszú parancslistákat gépelünk be (vagy sok felhasználói és gépnevet), akkor használjuk a sudo Alias változóit, amelyeket a sudo beállítások tetszőleges sorában használhatunk a megfelelő bejegyzések helyett:

```
User_Alias ADMINS=rob,jim,david
User_Alias WEBMASTERS=peter,nancy

Runas_Alias DAEMONS=bind,www,smmsp,ircd

Host_Alias
➥ WEBSERVERS=www.oreillynet.com,www.oreilly.com,
➥ www.perl.com

Cmnd_Alias
➥ PROCS=/bin/kill,/bin/killall,/usr/bin/skill,
➥ /usr/bin/top
Cmnd_Alias APACHE=/usr/local/apache/bin/apachectl

WEBMASTERS WEBSERVERS=(www) APACHE
ADMINS ALL=(DAEMONS) ALL
```

A felhasználó helyett csoportot is lehetséges megadni, hogy a csoportba tartozó összes felhasználó végrehajthassa a parancsokat. A csoportnév elé % (százalékjel) kerül:

```
%wwwadmin WEBSERVERS=(www) APACHE
```

Így minden felhasználó, aki a *wwwadmin* csoport tagja, kiadhatja az apachectl parancsot bármelyik webkiszolgálón.

Nagyon hasznos szolgáltatás a NOPASSWD: kapcsoló. Ezzel megadható, hogy a felhasználónak ne kelljen a jelszavát beírni a parancs végrehajtásához:

```
rob ALL=(ALL) NOPASSWD: PROCS
```

Ez megengedi rob felhasználónak, hogy bármelyik gépen, tetszőleges felhasználóként futtassa a *kill*, *killall*, *skill* és a *top* programot.

Végül a sudo jól használható a su helyett a rendszerindító fájlokban:

```
(cd /usr/local/mysql;
➥ sudo -u mysql ./bin/safe_mysqld &)

sudo -u www /usr/local/apache/bin/apachectl start
```

Ahhoz, hogy ez rendszerindításkor működjön, szükség van az alapértelmezett `root ALL=(ALL) ALL` sorra.

A sudo használatával kapcsolatban ugyanazok a megfontolások érvényesek, mint a *setuid* programoknál. Ha engedélyezzük a felhasználói beavatkozással futó programok (például szerkesztők), vagy bármilyen programnyelv értelmezőjének vagy fordítójának futtatását, fel kell tételeznünk, hogy a sudo felhasználó képes futtatható állományokat végrehajtani az adott felhasználói névvel. Ez azonban legtöbbször nem gond, és mindenképpen jobb, mint szükségtelenül rendszergazdai jogosultságokat adni.

## #13 Makefile használata rendszergazdai feladatok önműködő végrehajtására

### A Makefile mindent gyorsabban és könnyebben tesz (nem csak a C fordítást)

Valószínűleg mindenki ismeri a make programot, mint a programfordítás (legtöbbször a *gcc*-vel együtt alkalmazott) eszközét. Azonban nem sokan gondolnak rá, hogy a make – mivel nyilvántartja az állományok módosításának időpontjait – jól használható mindenféle frissítési feladat ellátására, amikor tetszőleges fájlok megváltoznak.

Következzen egy `Makefile`, ami a *Sendmail* beállításfájlok karbantartására szolgál:

```
M4= m4
CFDIR= /usr/src/sendmail-8.12.5/cf
CHMOD= chmod
ROMODE= 444
RM= rm -f
```

```
.SUFFIXES: .mc .cf

all: virtusers.db aliases.db access.db

access.db: access.txt
makemap -v hash access < access.txt

aliases.db: aliases
[TAB]newaliases

virtusers.db: virtusers.txt
makemap -v hash virtusers < virtusers.txt

.mc.cf:
$(RM) $@
$(M4) ${ CFDIR} /m4/cf.m4 $*.mc > $@ ||
➥ ( $(RM) $@ && exit 1 )
$(CHMOD) $(ROMODE) $@
```

Egy ilyen */etc/mail/Makefile* biztosítja, hogy ne kelljen emlékeznünk a *newaliases* futtatására minden alkalommal, amikor változtatunk a *Sendmail* másodnevek beállításán, vagy a `makemap` parancs helyes formájára, amikor a virtuális tartományok vagy a hozzáférések beállításait változtatjuk. Mindenekelőtt, amikor a felső szintű *mc* beállításfájlt változtatjuk (ugye mindenki az *mc* állományt használja, és nem a *sendmail.cf* fájlt piszkálja?), létrehozza az új *.cf* állományt, és mindehhez csak a `make` parancsot kell beírni. Mivel a `make` nyilvántartja a legutóbbi futtatás óta változtatott fájlokat, gondoskodik róla, hogy csak az készüljön el újra, amit tényleg meg kell változtatni.

Álljon itt még egy példa, ami Apache beállításállományokat helyez el egy másik kiszolgálón (például egy olyan osztott Apache telepítésnél, amilyet a [#99] „Terhelésmegosztás Apache RewriteMappel" trükk mutat be). Helyezzük a következő fájlt a */usr/local/apache/conf* könyvtárba:

```
#
# Makefile to push *.conf to the slave, as needed.
#
SLAVE= www2.oreillynet.com
APACHE= /usr/local/apache
```

```
RM= /bin/rm
TOUCH= /bin/touch
SSH= /usr/local/bin/ssh
SCP= /usr/local/bin/scp

.SUFFIXES: .conf .ts

all: test restart sites.ts globals.ts httpd.ts

configtest: test

test:
@echo -n "Testing Apache configuration: "
@$(APACHE)/bin/apachectl configtest

restart:
$(APACHE)/bin/apachectl restart

.conf.ts:
@$(RM) -f $@
@$(SCP) $*.conf $(SLAVE):$(APACHE)/conf
@$(SSH) $(SLAVE) $(APACHE)/bin/apachectl restart
@$(TOUCH) $@
```

Az a példa egy kicsit cselesebb, mert nem hozunk létre új állományokat, ezért a `make` nehezen tudná eldönteni, hogy megváltoztak-e a beállításfájlok. Üres *.ts* (időbélyeg, *TimeStamp*) fájlok létrehozásával csapjuk be a `make` programot, ezek csak a legutóbbi frissítés időpontjának tárolására szolgálnak. Ha a valódi beállításfájlok (*httpd.conf*, *sites.conf* vagy a *globals.conf*) változtak, először is az `apachectl configtest` parancsot futtatjuk, hogy ellenőrizzük, érvényesek-e a beállítások. Ha minden rendben, akkor a helyi Apache újraindítása következik, valamint a frissített állományok átmásolása a másik kiszolgálóra és az *Apache* újraindítása a másik kiszolgálón. Végül a *touch* frissíti a megfelelő *.ts* időbélyegeket, hogy a *.conf* fájlok a következő módosításig ne dolgozzák fel újra őket.

Ez sok gépelést takarít meg és lényegesen gyorsabb (és biztonságosabb), mint minden alkalommal saját kezűleg elvégezni mindezt.

# #14 Tartománynév-keresés nyers erőből

## Találjuk meg a tartománynevet, amit be szeretnénk jegyezni

Seregnyi segédeszköz érhető el a hálón, amivel megkereshetjük, hogy bejegyezhető-e még kedvenc tartománynevünk, és ha nem, ki foglalta le. Ezek általában webes alkalmazások és egyszerre néhány kérdés feltevését teszik lehetővé (és gyakran tesznek butácska javaslatokat hasonló nevekre, ha a keresett tartománynév már foglalt).

Ha nem annyira egy meghatározott tartománynévre vagyunk kíváncsiak, inkább egy adott mintának megfelelő nevet keresünk, miért ne bíznánk a munkát a parancssorra? Tegyük föl, hogy az összes „st" végű angol szót keressük:

```
cat /usr/share/dict/words | grep 'st$' |
➥ sed 's/st$/.st/' | \
while read i; do \
(whois $i | grep -q '^No entries found')
➥ && echo $i; sleep 60; \
done | tee list_of_st_domains.txt
```

Jó tanácsként elmondható: ha nem szeretnénk, hogy a szolgáltatónk kitiltson bennünket, akkor a `sleep 60`-at hagyjuk benne.

Ez a példa az összes olyan szót kilistázza szép sorban, amit még nem jegyeztek be az st felsőszintű tartomány karbantartójánál (aki nem más, mint el Republica Democratica de Sáo Tomé e Príncipe). Megkeresi a szótárban az st végű szavakat, és megpróbálja lekérdezni mindegyik szó *whois* bejegyzését egyenként, 60 másodperces időközökkel. Minden be nem jegyzett nevet a *list_of_st_domains* nevű állományba ment, és menet közben azt is mutatja hogyan halad. Cseréljük az st végződést másra (például us vagy to), és bármelyik felsőszintű tartomány neveit végigkérdezhetjük izomból.

Egyesek úgy érzik, hogy a tartománynevek zöld rétjén zajló bekerítések az internetet szinte vállalati gettóvá alakítják, de én nem értek ezzel egyet. Tulajdonképpen az egész helyzetet elég mókásnak találom.

# #15 Ki falja fel a merevlemezt?

## Keressük meg gyorsan a bűnöst egy ügyes héjprogrammal!

Mindig szombaton késő este történik. Az ember riasztást kap, mert az egyik kiszolgálón (valószínűleg levélkiszolgálón) egy lemezrész kezd vészesen megtelni.

A `df` megmutatja, mi maradt:

```
rob@magic:~$ df
Filesystem 1k-blocks Used Available Use% Mounted on
/dev/sda1 7040696 1813680 4863600 27% /
/dev/sda2 17496684 13197760 3410132 79% /home
/dev/sdb1 8388608 8360723 27885 100%
/var/spool/mail
```

De azt eddig is tudtuk, hogy a levelek tárhelye megtelt (ezért volt a riasztás, ami elrabolta egy amúgy kellemes, levélkiszolgáló nélküli esténket). Hogyan található meg gyorsan a lemezen elterpeszkedő felhasználó?

Ezt az egysorost érdemes felvenni a *.profile* fájlunkba:

```
alias ducks='du -cks * | sort -rn | head -11'
```

Ha ez a helyére került, a bármelyik könyvtárban kiadott `ducks` parancs megmutatja a teljes helyfoglalást és a tíz legtöbb helyet felemésztő felhasználót, csökkenő sorrendben. Végighalad az alkönyvtárakon, ami nagyon jól jön (de sok időt vehet igénybe egy erősen terhelt kiszolgálón vagy egy nagyon sok alkönyvtárat és fájlt tartalmazó könyvtárban). Járjunk a dolog végére:

```
rob@magic:~$ cd /var/spool/mail
rob@magic:/var/spool/mail$ ducks
8388608 total
1537216 rob
55120 phil
48800 raw
43175 hagbard
36804 mal
```

```
30439 eris
30212 ferris
26042 nick
22464 rachael
22412 valis
```

Hoppá! Úgy tűnik, elérkezett a levek tárhelyének végórája. Nem nagyságrendekkel több levelem van, mint bárki másnak? Tennem kell valamit, mondjuk, elsikkaszthatok egy új merevlemezt és átköltöztetem a */var/spool/mail* lemezrészt.

Mivel ez a parancs az alkönyvtárakat is végignézi, arra is jó, hogy rendszeres időközönként kimutatást készítsen a */home* könyvtárbeli helyfoglalásról:

```
root@magic:/home# ducks
[ pár másodperc múlva ]
13197880 total
2266480 ferris
1877064 valis
1692660 hagbard
1338992 raw
1137024 nick
1001576 rob
925620 phil
870552 shared
607740 mal
564628 eris
```

## #16 A /proc szépségei

### Nézzük meg közvetlenül a rendszermag folyamattábláját és a rendszer változóit

A */proc* fájlrendszer a rendszermag élő folyamattáblájának megjelenítése. A */proc* alatti fájlok és könyvtárak kezelése során megismerhető (és megváltoztatható) a futó rendszer számos jellemzője. Ne feledjük azonban, hogy rendszergazdaként bütykölni a */proc* alatt különösen veszélyes lehet, mivel a rendszergazdának jogában áll szinte bármit

felülírnia a folyamattáblában. Egyetlen elgépelt átirányítás, és a Linux engedelmesen lerombolja az egész rendszermag-memóriát, és még csak annyit sem mond: „Viszlát, kösz a kcore-okat!".

Következzen néhány érdekes példa arra, mi mindenre jó a */proc*. Ezekben a példákban feltételezzük, hogy a rendszermag viszonylag friss (2.4.18 vagy valami ilyesmi) és rendszergazdai jogosultságokkal rendelkezünk. Ha nem, akkor csak a saját felhasználói folyamatainkat tudjuk megnézni és módosítani.

Először egy kis terhelésű gépet nézzünk meg:

```
root@catlin:/proc# ls
1/ 204/ 227/ 37/ bus/ hermes/
➥ loadavg scsi/ version
1039/ 212/ 228/ 4/ cmdline ide/ locks self@
1064/ 217/ 229/ 5/
➥ cpuinfo interrupts meminfo slabinfo
1078/ 220/ 230/ 6/ devices iomem misc stat
194/ 222/ 231/ 698/ dma ioports modules swaps
197/ 223/ 232/ 7/ driver/ irq/ mounts sys/
2/ 224/ 233/ 826/ execdomains kcore net/ sysvipc/
200/ 225/ 254/ 827/
➥ filesystems kmsg partitions tty/
202/ 226/ 3/ apm fs/ ksyms pci uptime
```

A számokkal megnevezett könyvtárak a gépen futó összes folyamatról tárolnak adatokat. A szám a folyamatazonosítónak felel meg. A többi fájl és könyvtár a meghajtókat, számlálókat és a futó rendszermag más részleteit takarja. A felület éppen úgy működik, mint minden más fájl vagy eszköz a rendszerben; minden bejegyzés ugyanolyan módon írható és olvasható, mint a többi fájl. Tegyük fel, hogy meg akarjuk tudni, jelenleg a rendszermag melyik változata van betöltve:

```
root@catlin:/proc# cat version
Linux version 2.4.18 (root@catlin)
➥ (gcc version 2.95.3 20010315 (release))
➥ #2 Sat Jun 22 19:01:17 PDT 2002
```

Természetesen ennek nagy részét úgy is megtudhattuk volna, ha az `uname -a` parancsot használjuk, de így elkerüljük a közvetítőt (és tulajdonképpen azt tesszük, amit az `uname` is csinál).

Érdekelne, hogy mennyi RAM áll rendelkezésre? Nézzük meg a `kcore`-t:

```
root@catlin:/proc# ls -l kcore
-r-------- 1 root root 201330688
➥ Aug 28 21:39 kcore
```

Úgy látszik, 192 MB memória van a gépben (201330688/1024/1024 ≈ 192, többé-kevésbé). Figyeljük meg a korlátozott hozzáférési jogosultságokat. Ilyen módon védekezik a rendszer az ellen, hogy bárki közvetlenül olvashassa a memóriát. De a rendszergazda azt tehet, amit akar, így semmi akadálya, hogy a `grep` vagy a `strings` segítségével érdekes dolgokat mazsolázzunk ki a `kcore`-ból. Ez nem más, mint a rendszermemória, az adatok itt kerülnek gyorstárazásra addig, amíg felül nem íródnak (lehetővé téve elveszett adatok visszakeresését, vagy csúf szándékú felhasználók nyomon követését, akik olyasmit tesznek, amit nem volna szabad). Igazából a `kcore` mélyén tapogatózni nem túl szórakoztató elfoglaltság, és általában csak mélységes kétségbeesés (vagy nagy unalom) vihet rá valakit.

Még néhány figyelemreméltó fájl a rendszer állapotának megfigyelésére:

```
root@catlin:/proc# cat interrupts
CPU0
0: 34302408 XT-PIC timer
1: 2 XT-PIC keyboard
2: 0 XT-PIC cascade
3: 289891 XT-PIC orinoco_cs
8: 1 XT-PIC rtc
9: 13933886 XT-PIC eth0
10: 25581 XT-PIC BusLogic BT-958
14: 301982 XT-PIC ide0
NMI: 0
ERR: 0
```

A rendszer itt tartja nyilván az összes eddig használt megszakítást, a megszakítások számát és a megszakítást kérő meghajtó nevét.

```
root@catlin:/proc# cat partitions
major minor #blocks name

8 0 8971292 sda
8 1 8707198 sda1
8 2 257040 sda2
3 64 29316672 hdb
3 65 29310561 hdb1
```

Ez a rendszerindításkor felismert összes lemezrész listája a méretükkel együtt. A példában egy 9 GB méretű SCSI- és egy 30 GB-os IDE-lemez látható. Minden rendelkezésre álló lemezrész szerepel a listán, függetlenül attól, hogy be van-e fűzve (tehát könnyen ellenőrizhető, hogy vannak-e be nem fűzött lemezrészek a rendszerben).

Lépjünk tovább a rendszerjellemzőktől és nézzük meg az egyes folyamatok szerkezetét.

```
root@catlin:/proc# cd 1
root@catlin:/proc/1# ls -l
total 0
-r--r--r-- 1 root root 0 Aug 28 22:05 cmdline
lrwxrwxrwx 1 root root 0 Aug 28 22:05 cwd -> //
-r-------- 1 root root 0 Aug 28 22:05 environ
lrwxrwxrwx 1 root root 0 Aug 28 22:05 exe ->
    ➥ /sbin/init*
dr-x------ 2 root root 0 Aug 28 22:05 fd/
-r--r--r-- 1 root root 0 Aug 28 22:05 maps
-rw------- 1 root root 0 Aug 28 22:05 mem
lrwxrwxrwx 1 root root 0 Aug 28 22:05 root -> //
-r--r--r-- 1 root root 0 Aug 28 22:05 stat
-r--r--r-- 1 root root 0 Aug 28 22:05 statm
-r--r--r-- 1 root root 0 Aug 28 22:05 status
```

Ebben a könyvtárban három érdekes közvetett hivatkozás látható. A *cwd* a folyamat pillanatnyi munkakönyvtárára mutat (például a `cd /proc/3852/cwd` parancs abba a könyvtárba visz, ahonnan a 3852

azonosítójú folyamatot indították). Az *exe* hivatkozás a futtatott program teljes elérési útját tárja fel, a *root* pedig arra a könyvtárra mutat, ami az adott folyamat szerint a rendszergazdai könyvtár. A *root* majdnem mindig a /, kivéve, ha a program `chroot` paranccsal indult.

A *cmdline* és az *environ* fájl az eredetileg megadott parancssort és a folyamat összes környezeti változóját tartalmazza. Ezeket NULL karakterek választják el, ezért emberi fogyasztásra szánt formában a következőképpen íratható ki:

```
root@catlin:/proc/1# cat environ |tr '\ 0' '\ n'
HOME=/
TERM=linux
BOOT_IMAGE=catlin
```

(Vagy egy jobb példa, próbáljuk ki ugyanezt a */proc/self/environ* fájlon, ami a pillanatnyi folyamat környezetét tartalmazza):

```
root@catlin:/proc/1# cat /proc/self/environ |tr '\
0' '\ n'
PWD=/proc/1
HOSTNAME=catlin.nocat.net
MOZILLA_HOME=/usr/lib/netscape
ignoreeof=10
LS_OPTIONS= --color=auto -F -b -T 0
MANPATH=/usr/local/man:/usr/man:/usr/X11R6/man
LESSOPEN=|lesspipe.sh %s
PS1=\ u@\ h:\ w\ $
PS2=>
...
```

és így tovább. Ez remekül használható héjprogramokban (és egyéb programokban), amikor futó folyamatok bizonyos jellemzőit kell lekérdezni. De körültekintően járjunk el, és ne feledjük, hogy a szokásos jogokkal rendelkező felhasználók általában csak a saját folyamataik adataihoz férhetnek hozzá.

Végül következzen egy gyakorlati példa a */proc* használatára. A `ps` kimenetét az élő folyamattáblával összevetve látható, hogy a kettő azonos-e:

```
# ls -d /proc/* |rgrep [0-9]|wc -l; ps ax |wc -l
```

Ezzel a gyors szúrópróbával megkapjuk a futó folyamatok számát és a `ps` által megjelenített folyamatok számát. Sok betörőkészlet telepít megpiszkált `ps` programot, ami lehetővé teszi, hogy a rosszfiúk rejtve futtassanak programokat (egyszerűen azzal, hogy a `ps` ezeket nem jeleníti meg). El lehet bújni a `ps` elől, de sokkal nehezebb elbújni a */proc* elől. Ha a második szám lényegesen nagyobb, mint az első (különösen, ha több egymást követő próba is ezt mutatja), akkor nem árt további vizsgálat céljából leválasztani a gépet a hálózatról. Lehetőleg minél előbb!

## #17 Folyamatok közvetett kezelése a procps segítségével

### Küldjünk vezérlőjeleket a folyamatoknak (a folyamatazonosító helyett) név, terminál vagy felhasználói név alapján

Ha gyakran adjuk ki a `ps awux |grep` *`valami`* parancsot csak azért, hogy megtudjuk egy folyamat azonosítóját a `kill` parancs kiadásához, akkor itt az ideje megismerkedni néhány korszerű folyamatkezelő csomaggal.

Valószínűleg a *procps* a legjobban ismert folyamatkezelő csomag, ez tartalmazza a mindenhol elterjedt `top` program linuxos változatát is. A `top` olyan hihetetlenül hasznos és rugalmas segédeszköz, hogy külön is tárgyaljuk. Bővebbet a [#58] „Rendszererőforrások felügyelete a `top`-pal" részben tudhatunk meg róla.

A *procps* ügyes eszközei közé tartozik még az `skill`, amivel név, felhasználói név vagy folyamatazonosító alapján küldhetünk vezérlőjeleket a folyamatoknak, és az `snice`, ami ugyanilyen módon használható a `renice` helyett.

Például befagyasztható a `pts/2` terminál:

```
# skill -STOP pts/2
```

A következőképpen engedhetjük ki újra a tetszhalál szorításából:

```
# skill -CONT pts/2
```

Vagy módosítsuk a *luser* felhasználó minden folyamatának *nice* szintjét 5-re:

```
# snice +5 luser
```

A `pkill` hasonló az `skill` programhoz, de kötöttebb formájú. Nem próbálja meg kitalálni, hogy felhasználói névre, folyamatnévre vagy terminálra utal a parancs – ezt kapcsolókkal kell meghatároznunk. Például a következő két parancs ugyanazt csinálja:

```
# skill -KILL rob bash
# pkill -KILL -u rob bash
```

A `pkill` egy kicsit több gépelésbe kerül, de biztosan egyértelmű (például, ha egy felhasználó és egy folyamat neve véletlenül megegyezne).

A `pgrep` a `pkill` formájához hasonló, de ahelyett, hogy jelet küldene, csak kiírja a mintának megfelelő folyamat(ok) azonosítóját a szabályos kimenetre:

```
$ pgrep httpd
3211
3212
3213
3214
3215
3216
```

Végül a *vmstat* szép, könnyen feldolgozható formában ad pillanatfelvételt a memória- és processzorhasználatról:

```
$ vmstat
procs memory swap io system cpu
r b w swpd free buff cache si so bi bo in cs us sy id
0 0 0 5676 6716 35804 58940 0 0 9 9 7 9 0 0 29
```

Ha szeretnénk megfigyelni hogyan változik a terhelés, adjunk meg egy számot a parancssorban. A szám másodpercben megadja, mennyi idő teljen el két mérési eredmény megjelenítése között.

A kevésbé ismert *procps* segédeszközök használatát megtanulva rengeteg gépelés takarítható meg, na és sok szitkozódás is. Az `skill` vagy a `pkill` használatával megelőzhető, hogy a `kill` parancssorba véletlenül rossz folyamatazonosítót írjunk be, és váratlanul lelőjük az `sshd`-t, szegény fejünkre vonva sok dühös felhasználó átkait.

### Kapcsolódó anyagok

- `ftp://people.redhat.com/johnsonm/procps`
- [#58] Rendszererőforrások felügyelete a `top`-pal
- `man pkill`, `pgrep`, `skill`, `snice` és `vmstat`

## #18 Rendszererőforrások felügyelete az egyes folyamatok szintjén

### Előzzük meg, hogy a felhasználók lenyúlják a rendszer összes erőforrását

Elképzelhető, hogy egyetlen felhasználó akár szándékosan, akár véletlenül a rendszer összes erőforrását lefoglalja, gyenge teljesítményt vagy éppen rendszerleállást idézve elő. Az erőforrások felemésztésének megakadályozására az egyik – gyakran figyelmen kívül hagyott – kiváló eszköz a *bash* `ulimit` beállítása.

Az `ulimit -f` segítségével megelőzhető, hogy egy folyamat (vagy bármely leszármazottja) óriási állományokat hozzon létre (a legnagyobb fájlméretet kilobájtban adjuk meg):

```
rob@catlin:/tmp$ ulimit -f 100
rob@catlin:/tmp$
➥ yes 'Spam spam spam spam SPAM!' > spam.txt
File size limit exceeded
rob@catlin:/tmp$ ls -l spam.txt
-rw-r--r-- 1 rob users 102400 Sep 4 17:05 spam.txt
rob@catlin:/tmp$
```

A felhasználók csökkenthetik a saját méretkorlátjukat, de nem növelhetik (mint a `nice` és a `renice` esetében). Ez azt jelenti, hogy a */etc/profile* állományban megadott `ulimit` beállításokat később csak a rendszergazda növelheti:

```
rob@catlin:/tmp$ ulimit -f unlimited
bash: ulimit: cannot modify limit:
➥ Operation not permitted
```

Megjegyzem, semmi nem akadályozza meg a felhasználókat abban, hogy rengeteg fájlt hozzanak létre, mindegyiket a legnagyobb engedélyezett mérettel. Az ilyen hozzáállású felhasználót kísérjük egy eldugott helyiségbe és mutassuk be neki a legújabb LART (magyarul: LFS) eszközünket. Esetleg megfontolhatjuk tárkorlát (quota) bevezetését (bár ezt legtöbbször nem a legszórakoztatóbb megvalósítani, lehet, hogy az egyszerű meggyőzés módszerével is megoldható a helyzet).

Az `ulimit` hasonló módon az egy felhasználó által indítható folyamatok számát is korlátozhatja:

```
rob@catlin:~$ cat > lots-o-procs
#!/bin/bash
export RUN=$((RUN + 1))
echo $RUN...
$0
^D
rob@catlin:~$ ulimit -u 10
rob@catlin:~$ ./lots-o-procs
1...
2...
3...
4...
5...
6...
7...
8...
9...
./lots-o-procs: fork: Resource temporarily
unavailable
rob@catlin:~$
```

Ezzel a felhasználó összes terminálon (és háttérben) indított folyamatainak számát is szabályozzuk. Ennek így kell lennie, mivel amint egy folyamatot elágaztatunk, az függetleníti magát a vezérlőterminálról. (Vajon akkor hogyan számolhatnánk külön valamelyik héjhoz?)

Egy további nagyon hasznos `ulimit` kapcsoló a `-v`, a virtuális memória legnagyobb mérete. Amikor ezt a határt elérjük, a folyamatok a *„Segmentation fault"* hibaüzenettel leállnak (ami nem túl elegáns, de megelőzi, hogy a rendszer lefagyjon, amikor kifogy a fizikai és lapozómemóriából. Ha egy rendkívül rosszul megírt program vígan burjánzik a memóriában (például az Apache, a `mod_perl` vagy egy rosszul megírt CGI program), akkor „vészfék" gyanánt használhatjuk az `ulimit` beállítást, amíg a hiba okát megkeressük. Ismét kilobájtban adhatjuk meg a korlátot.

A rendelkezésre álló `ulimit` beállítások a `-a` kapcsolóval tekinthetők meg:

```
rob@catlin:~$ ulimit -a
core file size (blocks) 0
data seg size (kbytes) unlimited
file size (blocks) unlimited
max locked memory (kbytes) unlimited
max memory size (kbytes) unlimited
open files 1024
pipe size (512 bytes) 8
stack size (kbytes) 8192
cpu time (seconds) unlimited
max user processes 1536
virtual memory (kbytes) unlimited
```

Láthatjuk, hogy az egész rendszerre érvényes szigorú beállítások híján a felhasználói folyamatok elég nagyra duzzadhatnak. A *tcsh* héjban ugyanerre a célra a `limit` parancs szolgál:

```
rob@catlin:~> limit
cputime unlimited
filesize unlimited
datasize unlimited
stacksize 8192 kbytes
coredumpsize 0 kbytes
memoryuse unlimited
descriptors 1024
memorylocked unlimited
maxproc 1536
openfiles 1024
```

Az egész rendszerre kiterjedő korlátozások bevezetése drákói megoldásnak tűnhet, de sokkal jobb, mint amikor egy felhasználói program ámokfutása miatt az egész rendszer zuhanórepülésbe kezd.

## #19 Nagytakarítás a megszüntetett felhasználók után

### Győződjünk meg róla, hogy rendesen bezártuk-e a kaput a távozó felhasználó után

Ez is megtörténhet. Változnak a tervek, a cég más területre összpontosít, vagy mindig akadnak olyanok, akik továbbállnak. Egyszer minden rendszergazdának el kell takarítania a parancssori hozzáférést olyasvalaki után, aki elhagyta a céget, saját elhatározásából vagy másképp.

Személy szerint azon a véleményen vagyok, hogy azok a felhasználók, akikben kezdettől fogva nem bíznak meg, egy napon megbízhatatlannak fognak bizonyulni. (Az is igaz, hogy még soha nem kellett nyilvános parancssoros hozzáférést biztosító kiszolgálót üzemeltetnem egy internetszolgáltatónál.) Mindenesetre aki kezdettől fogva bizalmi viszonyt épít ki a felhasználókkal, az fontos lépést tett afelé, hogy később is nyugodtan tudjon aludni éjszakánként.

Amikor eljön az ideje, hogy egy felhasználó hozzáférését lezárjuk, leghelyesebb, ha módszeresen járunk el. Ne éljünk abban a hitben, hogy egy `passwd -l` kiadása után a felhasználó már nem képes hozzáférni a géphez. Tegyük fel, hogy egy *luser* nevű felhasználó után akarjuk bezárni a kaput. Néhány kézenfekvő (és néhány kevésbé kézenfekvő) dolog, amit a nagytakarítás során meg kell tennünk:

```
passwd -l luser
```

A felhasználó jelszavát befagyasztani, természetesen jó kiindulópont.

```
chsh -s /bin/true luser
```

Ez is népszerű lépés, olyan programra cseréljük a felhasználói héját, ami rögtön kilép. Ez általában megakadályozza, hogy a felhasználó parancssoros hozzáféréshez jusson a kiszolgálón. De vigyázzunk, ha

a gépen fut az `sshd` és a távoli RSA és DSA kulcsú azonosítást engedélyeztük, akkor *luser* még mindig továbbíthat kapukat bármilyen gépre, amit a kiszolgálónk el tud érni! Ezt a következő paranccsal teheti meg:

```
luser@evil:~$ ssh -f -N
➥ -L8000:sajat.intranet.kiszolgalo.com:80
➥ regi.kiszolgalo.com
```

A *luser* a helyi 8000 kaput helyi hálózati kiszolgálónk `http` kapujára továbbította. Ezt azért teheti meg, mert nem jelszóval azonosítja magát (RSA kulcsot használ) és nem próbál programot futtatni a *regi.kiszolgálo.com* nevű gépen (a `-N` kapcsolót ad meg).

Nyilvánvaló, hogy el kell távolítani a *~/lusesr/.ssh/authorized_keys** állományokat, ilyen módon akadályozva meg, hogy a *luser* az `ssh` kulcsait használhassa. Nézzük meg a következő állományokat is:

```
~luser/.shosts
~luser/.rhosts
```

Ennek többnyire csak akkor van jelentősége, ha az `rsh` szolgáltatást futtatjuk, vagy ha engedélyeztük ezt a szolgáltatást az `ssh`-ban. De soha nem tudhatjuk, később egy másik rendszergazda nem engedélyezi-e a *.shosts* vagy *.rhosts* szolgáltatást, úgyhogy leghelyesebb most eltávolítani a fájlokat, ha léteznek.

Volt-e `sudo` jogosultsága a *luser* felhasználónak? Ellenőrizzük a `visudo` segítségével.

Hogyan állunk a `cron` vagy `at` folyamatokkal?

```
crontab -u luser -e
atq
```

Ha már itt tartunk, futtat-e valamilyen programot a *luser?*

```
ps awux |grep -i ^luser
```

Vagy a [#17] „Folyamatok közvetett kezelése a procps segítségével" részben olvasottak alapján:

```
skill -KILL luser
```

Tudott-e a *luser* CGI programokat futtatni a saját könyvtárából (vagy máshonnan)?

```
find ~luser/public_html/ -perm +111
```

És a PHP vagy hasonló beágyazott parancsnyelvek?

```
find ~luser ~public_html/ -name '*.php*'
```

Beállított-e a *luser* levéltovábbítást ? A továbbítóállomány gyakran tetszőleges program futtatására használható.

```
less ~luser/.forward
grep -C luser /etc/mail/aliases
```

Végül vannak-e a *lusernek* gyanús helyen lévő állományai?

```
find / -user luser > ~root/luser-files.report
```

Az egyik biztos (és gyors) módszer, amivel meggyőződhetünk róla, hogy a *luser* felhasználó összes beállításfájlja érvénytelenítve van, az `mv /home/lluser /home/luser.removed` parancs. Ezzel érintetlenül hagyható a felhasználó könyvtárának tartalma anélkül, hogy aggódnunk kellene, nem hagytunk-e figyelmen kívül valamilyen behatolási pontot. Így azonban az ártalmatlan *.forward* állományt is működésképtelenné tesszük (csakúgy, mint a *~luser/public_html* könyvtárat és minden más nyilvánosan hozzáférhető adatot, amit a *luser* a könyvtárában tartott). Ha ezt a módszert választjuk, mindenképpen orvosoljuk ezt a hibát (például helyezzük el a megfelelő bejegyzést a rendszer *aliases* állományában, és másoljuk vissza a *public_html* megtisztított változatát a */home/luser/public_html/* könyvtárba).

Nézzünk utána, volt-e a *luser* felhasználónak hozzáférési jogosultsága beállítási- vagy rendszerfájlokhoz, különös tekintettel az emelt szintű jogosultságokkal futó szolgáltatásokra. Még egy olyan ártalmatlannak tűnő apróság, mint egy felhasználótól származó Apache beállításfájl is felhasználható arra, hogy parancssori hozzáférést tegyen lehetővé.

Ez a felsorolás még korántsem teljes, de érzékelteti, hogy a hozzáférés visszavonása sokkal többől áll, mint a felhasználó jelszavának lezárása. Ha a felhasználónak valaha is rendszergazdai hozzáférése volt, akkor lehet találgatni. A trójai faló programoktól kezdve a láthatatlan magmodulokon át a rendszergazdai jelszó egyszerű megváltoztatásáig bárhogyan bejuthat a rendszerbe.

Ismerjük meg a parancssori hozzáféréssel rendelkező felhasználókat még jóval azelőtt, hogy búcsút kellene mondanunk nekik. Ez a munka elég nehéz akkor is, ha nem kell azon rágódnunk, hogy kiben bízhatunk meg. Léteznek olyan programok, mint például a `screen`, amely segítségével a törlés után is akár szándékos kárt is okozhat.

## #20 A fölösleges meghajtók eltávolítása a rendszermagból

### Szabjuk át a rendszermagot a gyorsabb indítás és a hosszú távú üzembiztonság érdekében!

A Linux egészen elképesztően sokfajta kiépítésű számítógépen futhat. A legkülönfélébb részegységekhez érhető el támogatás, kezdve az alapvető fontosságú alkatrészeken (mint a merevlemezek és a RAM), a jóval egzotikusabb eszközökig (mint az USB lapolvasók és videodigitalizálók). A Linux-terjesztésekkel kapott rendszermag a teljesség (és biztonság) szem előtt tartásával készül, de valószínűleg valamennyit azért fel kell áldozni a hatékonyságból, mivel a lehető legtöbb eszköz támogatása a cél.

Figyeljük meg a rendszermag üzeneteit a rendszerindításkor. Azt fogjuk tapasztalni, hogy megpróbál mindenféle alkatrészt felderíteni (elsősorban a SCSI-vezérlőkre és ethernetkártyákra gondolhatunk), ami nincs a gépben. Ha az adott terjesztés elrejti a mag rendszerindítási üzeneteit, használjuk a `dmesg` parancsot (kimenetét a `less`-re irányítva), hogy a rendszer indulásakor létrehozott üzeneteket megnézzük.

Annak érdekében, hogy a rendszerindítás gyorsabb legyen, és kizárjuk annak a lehetőségét, hogy egy felesleges meghajtó esetleg a gépben lévő alkatrészekkel nem fér össze, nem árt a géphez szabni, hogy a rendszermag milyen meghajtókat próbáljon meg betölteni.

A rendszermag meghajtóinak használatával kapcsolatban kétfajta nézet alakult ki. Vannak, akik minden szükséges szolgáltatást a rendszermagba építenek, és elkerülik a betölthető magmodulok használatát. Mások kisebb rendszermagot állítanak össze, és rendszerindításkor töltik be a meghajtókat. Természetesen mindkét módszernek megvannak az előnyei és a hátrányai.

Például a monolit (betölthető modulok nélküli) rendszermaggal biztosan elindul a rendszer akkor is, ha a */lib/modules* könyvtárban lévő meghajtókkal történik valami. Egyes rendszergazdák olyan rendszermagot szeretnek építeni, amiből még a modulok támogatása is hiányzik, így nem fordulhat elő, hogy egy sötét szándékú ismeretlen trójai falóként működő eszközmeghajtót töltsön be. Másrészt viszont, amikor új alkatrészt adunk a géphez, annak támogatásához újra kell fordítanunk a rendszermagot.

A betölthető modulok óriási rugalmasságot biztosítanak az eszközmeghajtók betöltésében. A meghajtók betöltésének sorrendje módosítható, sőt kapcsolókat is megadhatunk a meghajtóknak, s mindezt újraindítás nélkül. A hátrány az, hogy a */lib/modules* könyvtárnak a modulok megfelelő változatát kell tartalmaznia, különben a rendszer nem tudja betölteni a meghajtókat. Új rendszermag fordításakor nem szabad megfeledkezni a `make modules_install` parancsról, és a modulkezelő segédprogramokat (`modprobe`, `lsmod`) is naprakészen kell tartanunk. A betölthető modulokkal fordított rendszermag akkor is segíthet, ha a monolit rendszermag túl nagy, és nem töltődik be (azzal, hogy egyes eszközmeghajtókat csak akkor tölt be, amikor a rendszer már elindult és befűzte a fájlrendszereket).

Függetlenül attól, hogy melyik módszert használjuk, olyan rendszermagra lesz szükségünk, ami csak a legszükségesebb szolgáltatásokat tartalmazza. Ez elsősorban annak az IDE, SCSI vagy más meghajtónak (vagy esetleg hajlékonylemeznek vagy éppen hálózati kártyának)

a támogatását jelenti, amiről a rendszer indul. Szükség lesz arra a fájlrendszerre is, amit a gyökérlemezrészen használunk (ez leginkább `ext2`, `ext3` vagy a *ReiserFS*). Figyeljünk rá, hogy a rendszermag annyi memóriát támogasson, amennyi a gépben van (lásd [#21] Nagy mennyiségű RAM használata). Válasszuk ki a gépünkhöz illő processzortípust, hogy a számára leghatékonyabbat kapcsoljuk be. Ha egynél több processzort tartalmaz a gép, mindenképpen SMP-rendszermagot építsünk.

A géphez szabott rendszermag fordításához először csomagoljuk ki a forráskódot olyan helyre, ami elég lesz neki (legyen például a */usr/local/src/linux*). Adjuk ki a `make menuconfig` (vagy X alatt a `make xconfig`) parancsot. Körültekintően válasszuk ki a meghajtókat (az `Y` a beépített, az `M` a modulba fordított meghajtókat jelenti). Ne feledjük, hogy kiszolgáló számára fordítunk rendszermagot; ne fordítsunk hozzá felesleges meghajtókat. Egy kiszolgálón valószínűleg nem kell hangkártya-támogatás, még akkor sem, ha az alaplapra van építve. És az USB? Ha nincs meghatározott, a kiszolgáló feladatához közvetlenül kapcsolódó USB-eszközünk, akkor ne telepítsünk USB-meghajtót. A nem használt alkatrészeket a BIOS menüben is érdemes letiltani, hogy rendszererőforrásokat szabadítsunk fel és csökkentsük az ütközések esélyét.

Miután kiválasztottuk a rendszermag lefordítandó részeit, a beállítások a rendszermag forrásfájának legfelső könyvtárában lévő *.config* állományba kerülnek. Ezt a fájlt mentsük is a későbbi használatra, mert nagyon megkönnyíti majd a dolgunkat a rendszermag frissítésekor (ha az új mag forrásfájának tetejére másoljuk, és a `make oldconfig` paranccsal fordítunk). Ezután fordítsuk az új rendszermagot (és a modulokat, ha vannak), telepítsük, és próbáljuk is ki. Feltétlenül próbáljuk ki az összes eszközt egy új rendszermag telepítése után, és figyeljünk oda a rendszerindítási üzenetekre. Ekkor célszerű megkeresni a váratlan hibaüzenetek és figyelmeztetések okát, nehogy a későbbiekben idézzenek elő hibát.

Végül, ha a rendszermagot hálózaton keresztül telepítjük, ne felejtsük el belefordítani a hálózati eszközök meghajtóit sem! Semmi nem hasonlítható ahhoz a kínhoz, mint amikor az ember közvetlenül

a `shutdown -r now` kiadása után ráébred, hogy a hálózati meghajtókat kifelejtette. Jó esetben fel lehet hívni valakit, akinek van konzolról hozzáférése a géphez (aki tapintatosan újraindítja a rendszert a régi maggal, és nem meséli el túl sok barátunknak a történteket).

Pontosan a géphez szabott rendszermagot építeni komoly feladat lehet, de szükséges ahhoz, hogy a kiszolgáló a lehető leghatékonyabban működjön. Ne szegje kedvünket, ha jó néhányszor újra kell próbálkoznunk a tökéletes rendszermag elkészítéséig; bőven megéri az erőfeszítést.

### Kapcsolódó anyagok

- Running Linux, negyedik kiadás (O'Reilly)
- [#21] Nagy mennyiségű RAM használata

## #21 Nagy mennyiségű RAM használata

### Tegyük elérhetővé a teljes memóriát a Linux számára

A Linux x86 alapú rendszereken képes akár 64 GB RAM kezelésére is, de a 960 MB fölötti memória jelenlétét közölnünk kell a rendszerrel.

Először is a rendszermagnak támogatnia kell a többletmemóriát. Az alapértelmezett beállításokkal a rendszermag általában 960 MB memóriát kezel. Ha ennél többet rakunk a gépbe, azt egyszerűen figyelmen kívül hagyja. (A leggyakoribb panasz az, hogy az 1 GB memóriából *free* csak 960 MB-ot mutat, pedig a BIOS-ellenőrzés 1024 MB-ot számolt.)

A rendszermag a rendelkezésre álló memóriát a *High Memory Support* beállítás (vagyis a `CONFIG_NOHIGHMEM` változó) alapján kezeli. A használt memóriamérettől függően módosítsuk a beállítást:

```
960 MB-ig: off
4 GB-ig: 4 GB
több mint 4 GB: 64 GB
```

Vegyük figyelembe, hogy a 64 GB beállítás csak az Intel Physical Address Extension (PAE) módot támogató processzorokkal működik. A rendszermag leírása szerint a Pentium Pro óta minden Intel processzor támogatja a PAE-módot, de a régebbi processzorokkal ez a beállítás nem működik (és a rendszermag nem indul el; ez az egyik oka annak, hogy alapértelmezés szerint nincs engedélyezve a PAE-mód). Végezzük el a szükséges beállításokat és fordítsuk újra a rendszermagot (lásd [#20] A fölösleges meghajtók eltávolítása a rendszermagból).

Amikor elkészítettük és telepítettük a rendszermagot, lehet, hogy a rendszerindító programmal közölnünk kell a memória méretét (ugyanis nem minden BIOS érzékeli pontosan a RAM méretét a rendszerindításkor). Ehhez használjuk a `mem=`*`rendszermag-kapcsoló`*-t a rendszerindító program beállításában. Tegyük fel, hogy a gépben 2 GB memória van.

Ha a LILO-val indítjuk a rendszert, a következő sort adjuk meg a */etc/lilo.conf* állományban:

```
append="mem=2048M"
```

A Grub beállításához a */etc/grub.conf* fájlba a következőt írjuk:

```
kernel /boot/vmlinuz-2.4.19 mem=2048M
```

A `loadlin` használata esetén a `loadlin` parancssorban adjuk meg:

```
c:\ loadlin c:\ kernel\ vmlinuz root=/dev/hda3
➥ ro mem=2048M
```

Bár elég ritkán találkozunk olyan kiszolgálóval, ami `loadlin`-nel indul, nem igaz? ;-)

## #22 hdparm: az IDE-meghajtók finomhangolása

### Hozzuk ki a lehető legtöbbet a merevlemezekből!

Aki legalább egy (E)IDE-meghajtóval rendelkező linuxos gépet használ, és még nem hallott a *hdparm* csomagról, az olvasson tovább.

A Linux-terjesztések alapértelmezés szerint nem használnak különleges beállításokat az IDE-illesztő és merevlemezek eléréséhez. Ez nagyon óvatos megközelítés, fő szempontja: az adatok biztonsága min-

denekelőtt. De amint azt már észrevehettük, a biztonsági beállítások szinte soha nem gyorsak. Ráadásul a sok adatot feldolgozó alkalmazások számára ismeretlen fogalom az „elég gyors".

Ha a legtöbbet akarjuk kihozni az IDE-részegységekből, vessünk egy pillantást a `hdparm(8)` parancsra. Nemcsak azt mutatja meg, hogy pillanatnyilag hogyan teljesítenek a merevlemezek, hanem tetszés szerint módosíthatjuk is a beállításaikat.

Meg kell jegyeznünk, hogy bizonyos körülmények között ezek a parancsok *váratlan adatvesztést eredményezhetnek!* Mindenki a saját felelősségére kísérletezzen! A legkevesebb, hogy készítsünk biztonsági másolatot és váltsunk egyfelhasználós módba, mielőtt tovább haladnánk.

Lássunk hozzá! Most, hogy egyfelhasználós módban vagyunk (erről már volt szó, lásd [#2] „A bejelentkezés megkerülése" trükköt), nézzük meg, hogyan teljesít az első IDE-merevlemez:

```
hdparm -Tt /dev/hda
```

Valami ehhez hasonlót kell látnunk:

```
/dev/hda:
Timing buffer-cache reads:
➥ 128 MB in 1.34 seconds =95.52 MB/sec
Timing buffered disk reads:
➥ 64 MB in 17.86 seconds = 3.58 MB/sec
```

Mit tudunk meg ebből? A `-T` a gyorstárrendszer (vagyis a memória, a processzor és a merevlemez gyorstár) sebességét méri. A `-t` a kérdéses lemez adatait méri, nem a gyorstárból, hanem a lemezről átvitt adatokkal. A kettő együtt, többször futtatva az egyfelhasználós módban, tájékoztat a merevlemezek sebességéről. (A következő számok Pentium II, 350 MHz, 128 MB RAM, EIDE HD gépről származnak; az eredmények különbözhetnek.)

Még ha különbözhetnek is az eredmények, 3,58 MB/s szánalmas teljesítmény a fenti gépen, és ha jól emlékszem, a merevlemez reklámjában valami 66 MB/s szerepelt! Most mi történt?

Nézzük meg közelebbről, hogyan kezeli a Linux ezt a merevlemezt:

```
# hdparm /dev/hda

/dev/hda:
multcount = 0 (off)
I/O support = 0 (default 16-bit)
unmaskirq = 0 (off)
using_dma = 0 (off)
keepsettings = 0 (off)
nowerr = 0 (off)
readonly = 0 (off)
readahead = 8 (on)
geometry = 1870/255/63, sectors = 30043440, start = 0
```

Ezek az alapbeállítások. Szép, biztonságos, de nem feltétlenül a legjobb. Mi ez a 16-bites üzemmód? Úgy tudtam, ez a 386-os óta kiment a divatból!

Ezekkel a beállításokkal szinte minden elképzelhető alkatrész munkára bírható, de mivel most egy kicsit többet szeretnénk munkára bírni, mint egy poros, nyolcéves 16-bites illesztőkártyát, vegyük sorra az érdekesebb beállítási lehetőségeket:

`multcount`
: Ez a „multiple sector count" rövidítése; azt adja meg, hogy a rendszer egy megszakítás alkalmával hány szektort olvas be a merevlemezről. Ezt szinte minden korszerű IDE-merevlemez támogatja. A súgó szerint: „amikor ez a szolgáltatás be van kapcsolva, a rendszer terhelése a lemezműveletek alatt jellemzően 30–50 százalékkal csökken". Sok gépen emellett öt és ötven százalék közötti adatátviteli sebességnövekedést is eredményez.

`I/O support`
: Ez szabályozza, hogy az adatok hogyan kerülnek a PCI buszról az illesztőre. Szinte minden korszerű illesztő támogatja a 3-as módot, vagyis 32-bites átvitelt *sync* jelsorral. Bizonyos illesztők a *sync* nélküli átvitelt is támogatják. Ezt beállítva szinte biztosan megkétszereződik az átviteli sebesség (lásd alább).

`unmaskirq`
: Ez a beállítás megengedi, hogy a Linux engedélyezzen más megszakításokat, miközben a merevlemez megszakítási kérelmét feldolgozza. Mit jelent ez? Lehetővé teszi, hogy a Linux más, megszakítással működő feladatokat (például hálózati adatátvitelt) végezzen, miközben arra vár, hogy a merevlemezről megérkezzenek az adatok. Ez csökkenti a rendszer válaszadási idejét, de vigyázzunk: nem minden alkatrész-összeállítás képes kezelni ezt a beállítást. Lásd a súgót.

`using_dma`
: A DMA cseles, ha be tudjuk állítani a csatolót és a merevlemezt DMA-módra, akkor tegyük meg. Mindazonáltal nem egy gépet láttam már lefagyni, miközben ezzel a beállítással nyúzták. Ismét csak: lásd a súgót.

Próbáljunk ki néhány gyors beállítást:

```
# hdparm -c3 -m16 /dev/hda

/dev/hda:
setting 32-bit I/O support flag to 3
setting multcount to 16
multcount = 16 (on)
I/O support = 3 (32-bit w/sync)
```

Nagyszerű! A 32 bit jól hangzik, és a többszektoros olvasás is jól jöhet. Futtassuk újra a mérést:

```
# hdparm -tT /dev/hda

/dev/hda:
Timing buffer-cache reads:
➥ 128 MB in 1.41 seconds =90.78 MB/sec
Timing buffered disk reads:
➥ 64 MB in 9.84 seconds = 6.50 MB/sec
```

Hmmm, majdnem kétszeres átviteli sebesség, és még hozzá sem fogtunk igazán! Lenyűgöző.

De várjunk csak: még nem engedélyeztük a megszakításokat, és nem használunk DMA-t, vagy legalább egy tisztességes PIO-módot! Ezekkel már nagyobb kockázatot vállalunk. A súgó említi az MDMA 2-módot, próbáljuk is ki:

```
# hdparm -X34 -d1 -u1 /dev/hda
```

Ezen a gépen, úgy látszik, sajnos ez nem használható (úgy lefagyott, mint egy Java-alkalmazást futtató NT). Ezért újraindítás után (megint egyfelhasználós módban) a következővel próbálkoztam:

```
# hdparm -X66 -d1 -u1 -m16 -c3 /dev/hda

/dev/hda:
setting 32-bit I/O support flag to 3
setting multcount to 16
setting unmaskirq to 1 (on)
setting using_dma to 1 (on)
setting xfermode to 66 (UltraDMA mode2)
multcount = 16 (on)
I/O support = 3 (32-bit w/sync)
unmaskirq = 1 (on)
using_dma = 1 (on)
```

Majd ellenőriztem:

```
# hdparm -tT /dev/hda

/dev/hda:
Timing buffer-cache reads:
➥ 128 MB in 1.43 seconds =89.51 MB/sec
Timing buffered disk reads:
➥ 64 MB in 3.18 seconds =20.13 MB/sec
```

20,13 MB/s. Messzire jutottunk a nevetséges 3,58-os értéktől.

Megfigyelhető, hogy újra megadtuk a `-m16` és a `-c3` kapcsolót. Ez azért szükséges, mert a *hdparm* beállítások újraindításkor elvesznek. Ha meggyőződtünk róla, hogy a rendszer üzembiztos az adott

beállításokkal, adjuk hozzá a fenti sort valamelyik */etc/rc.d/** parancsfájlhoz (lehetőleg az `fsck` futtatása után, mert kiterjedt állományrendszer-ellenőrzést végezni egy hibás módban üzemelő illesztővel ugyan jól használható módszer nagy mennyiségű entrópia létrehozására, de a rendszerfelügyelet során aligha alkalmazható. Legalábbis vigyorgás nélkül.)

Ha nem találjuk a *hdparm* programot a rendszerünkben (általában a */sbin* vagy a */usr/sbin* könyvtárban helyezkedik el), a forráskódot a `http://metalab.unc.edu/pub/Linux/system/hardware/` címről tölthetjük le.

# Változatkövetés

## 23–36. trükkök

Aki komolyan veszi a rendszerfelügyelet munkáját, valószínűleg számolatlan mennyiségű időt fordít rá, hogy a rendszer *pont úgy* működjön, ahogyan kell. Sajnos nem mindig lehet megmondani, hogy egy adott alkalmazás mikor „működik" és mikor „hibás" – a működőképességet ennél sokkal finomabb skálán mérik, valahol a fut és a leállt közötti szürke sávban helyezhető el. Még egy apró beállításmódosítás is olyan rafinált hatásokat válthat ki, amelyek napokig (vagy hetekig) nem láthatók.

Ebben lehet segítségünkre a változatkövetés. Az egyszerű biztonsági mentéseken kívül minden valamirevaló változatkövető rendszer számon tartja bármely fájl módosításait, valamint azt, hogy ki, mikor és miért módosította azt. Ezek az adatok leírhatatlan értéket képviselhetnek egy összetett rendszer esetében, főleg, ha több ember végzi a karbantartását. Ha a legfontosabb beállításfájlokat egy olyan rendszerben tároljuk, mint az RCS (Revision Control System) vagy CVS (Concurrent Versioning System), akkor a fájlok tetszőleges korábbi állapotához „vissza lehet tekerni", és az akkori és a jelenleg érvényes beállítások közötti különbségek elemezhetők.

Ennek a résznek az a célkitűzése, hogy az RCS és a CVS hatékony használatához szükséges tudnivalókat hasznos példaalkalmazásokkal együtt ismertesse. Az RCS rendkívül jól alkalmazható egyetlen gépen lévő állományok változatainak nyomon követésére, a CVS pedig kiterjedt tároló (archiving) és egyesítő (merging capabilities) képességekkel bír, és a hálózaton egy központi lerakóban (*repository*) tartja a változatokat. A CVS érthető módon sokkal összetettebb eszköz, mint az RCS. A 23–25. trükkök RCS gyorstalpaló tanfolyamot nyújtanak, míg a 26–36. trükkök megadják a kezdősebességet a CVS használatához, kezdve az egyszerű parancsoktól egészen a saját anonim CVS-lerakó kialakításáig.

Ez a fejezet természetesen feltételezi, hogy az olvasó számára nem jelent gondot a parancssoros fájlkezelés, még ha korábban nem is dolgozott RCS vagy CVS rendszerrel. Aki el szeretné mélyíteni CVS-ismereteit, annak Karl Fogel kitűnő *Open Source Development with CVS* című könyvét ajánlom (CoriolisOpen Press).

## #23 Bevezetés az RCS-be

### A rendszerfájlok és módosítások tárolását bízzuk az RCS-re

Az RCS változatkövető rendszer, arra használható, hogy beállításfájlok, parancsfájlok vagy bármilyen más fájlok különböző változatait tárolja. A CVS-től eltérően semmiféle támogatást nem tartalmaz a hálózati használathoz, mindent a helyi állományrendszeren tárol.

Az RCS minden változatot az *RCS/* nevű alkönyvtárban tart (a pillanatnyilag érvényes munkakönyvtárban). Új RCS-lerakó indításához először a könyvtárat hozzuk létre:

```
root@catlin:/etc# mkdir RCS
```

Mielőtt dolgozni kezdenénk vele, be kell jegyeznünk az állományt az új RCS-lerakóba. Legyen az állomány a *syslog.conf*:

```
root@catlin:/etc# ci -i syslog.conf
RCS/syslog.conf,v <-- syslog.conf
```

```
enter description, terminated with single '.'
➥ or end of file:
NOTE: This is NOT the log message!
>> Here's the syslog.conf for Catlin
initial revision: 1.1
done
```

Az első `ci -i` („*check in*" és „*initialize*") berakja és bejegyzi a fájlt, hogy az RCS rendelkezzen egy kiinduló változattal. Az RCS kér egy kezdeti leírást, azután eltávolítja a fájlt a pillanatnyi könyvtárból. Valószínűleg nem akarjuk túl sokáig ennyiben hagyni a dolgot, tehát miután bejegyeztük, kérjük ki a fájlt:

```
root@catlin:/etc# co syslog.conf
RCS/syslog.conf,v --> syslog.conf
revision 1.1
done
```

Ezzel visszamásoljuk a fájlt a könyvtárba, de még nem kezdhetünk el dolgozni rajta. Először kérjük ki lezárással, hogy mások ne írhassák át, amíg dolgozunk vele.

```
root@catlin:/etc# co -l syslog.conf
RCS/syslog.conf,v --> syslog.conf
revision 1.1 (locked)
done
```

Most már tetszés szerint szerkeszthetjük. Amikor végeztünk vele, rakjuk vissza és oldjuk föl a zárat a `-u`-val:

```
root@catlin:/etc# ci -u syslog.conf
syslog.conf,v <-- syslog.conf
new revision: 1.3; previous revision: 1.2
enter log message, terminated with single '.'
➥ or end of file:
>> Added remote logging to the new security log host
>> .
done
```

Mindenképpen használjunk beszédes naplóbejegyzéseket, mert *nem* fogunk emlékezni rá mit is értettünk a „néhány apró változtatás" alatt, ha fél év múlva fura működést tapasztalunk.

Ha gondunk akad a kikéréssel vagy a berakással, valószínűleg valamikor elfeledkeztünk a `-l` kapcsolóról (a `co` parancshoz) vagy a `-u` kapcsolóról (a `ci` parancshoz). Általában lehetőség nyílik az újrakezdésre, ha a fájlról készítünk egy másolatot, azután megint kikérjük a fájlt és visszamásoljuk. Tegyük fel, hogy épp most szerkesztettük a *syslog.conf* állományt:

```
root@catlin:/etc# ci -u syslog.conf
syslog.conf,v <-- syslog.conf
ci: syslog.conf,v: no lock set by root
root@catlin:/etc#
```

Ajaj, jobb lesz menteni és elölről kezdeni az egészet:

```
root@catlin:/etc# cp syslog.conf syslog.conf.backup
root@catlin:/etc# co -l syslog.conf
syslog.conf,v --> syslog.conf
revision 1.4 (locked)
done
root@catlin:/etc# cp syslog.conf.backup syslog.conf
root@catlin:/etc# ci -u syslog.conf
syslog.conf,v <-- syslog.conf
new revision: 1.5; previous revision: 1.4
enter log message, terminated with single '.'
➥ or end of file:
>> commented out the FIFO line
>> .
done
```

A következőkben az RCS további szolgáltatásait ismerhetjük meg.

# #24 Korábbi változatok kikérése az RCS-ből

## Az RCS mint mentőöv

Egyszer elkerülhetetlenül megesik, hogy az ember annyira tönkreteszi a dolgokat, hogy reménye sem marad a helyreállításra. Tegyük fel, hogy éppen most végeztünk bonyolult átalakításokat a *httpd.conf* állományon, és ezt a hibaüzenetet kapjuk, ami minden rendszergazda szívében félelmet kelt:

```
root@catlin:/usr/local/apache/conf#
➥ ../bin/apachectl restart
apachectl restart:
➥ httpd not running, trying to start
apachectl restart: httpd could not be started
```

Jaj, mit tettem! Ha a webkiszolgáló leállt, az emberek még a barátságos 404-es lapot sem láthatják, csak a „csatlakozás elutasítva" hibaüzenetet kapják. Ekkor kétségbeesetten áshatnánk le magunkat a *httpd.conf* legmélyére a `vi` szerkesztővel, hogy megtudjuk pontosan mit is tettünk (avagy mit tett az előző rendszergazda), ami ilyen büntetést érdemel.

De aki RCS-t használ, annak nem ilyen kilátástalan a helyzete, ugyanis az `rcsdiff` megmutatja, mi változott a legutóbbi és a mostani kikérés óta:

```
root@catlin:/usr/local/apache/conf#
➥ rcsdiff httpd.conf
===================================================
====================
RCS file: RCS/httpd.conf,v
retrieving revision 1.1
diff -r1.1 httpd.conf
458c458
< ErrorLog /usr/local/apache/logs/error_log
---
> ErrorLog :wq/usr/local/apache/logs/error_log
```

Na, ez az! Látható, hogy véletlenül bekerült egy `:wq` a 458. sorba. Nem mindig találjuk el az ESC billentyűt, igaz? Javítsuk ki és indítsuk újra az Apache-t, majd rakjuk el újra a fájlt.

De mit tegyünk, ha ennél messze ágazóbb változások történtek? Mondjuk, egy beállításfájlt alaposan átszabunk kedd reggel, és csütörtök délután kezdenek előrajzani a szörnyecskék. Hogyan térhetünk vissza az utolsó biztosan működő változathoz?

Egyszerűen: kérjük ki az adott változatot a `-r` kapcsolóval:

```
root@catlin:/etc# co -l -r1.2 syslog.conf
syslog.conf,v --> syslog.conf
revision 1.2
done
```

Presto, visszakerültünk abba az áldott állapotba, amit 1.2-es változatként raktunk el. Ne feledjük, az RCS esetében érdemes korán és gyakran menteni.

## #25 A módosítások követése: rsc2log

### Egy pillantással felmérhető, ki nyúlt a fájlokhoz és miért

Az RSC akkor mutatja meg igazi erejét, amikor arra van szükség, hogy egy állományt többen is megváltoztathassanak. Több rendszergazda együttműködése gyakran torkollik egymásra mutogatásba, ha valami rosszul működik. Ilyen estben nyilvánvaló, hogy valaki változtatott *valamit*, de senki sem akarja vállalni érte a felelősséget. Ha RCS-ben követjük a módosításokat, könnyű megmondani, hogy ki és mit változtatott:

```
root@www:/usr/local/apache/htdocs#
➥ rcs2log index.html
2002-08-14 rob <rob@mouse>

* index.html: meeting announcement
```

```
2002-07-30 rob <rob@mouse>

* index.html: gre.tunnel announcement

2002-07-12 sderle <sderle@mouse>

* index.html:
➥ added Marin and shuffled around a bit.

2002-07-10 rob <rob@mouse>

* index.html: meeting announcement + v0.81

2002-07-01 jim <jim@mouse>

* index.html: *** empty log message ***

2002-06-20 rob <rob@mouse>

* index.html: meeting reminder
```

Hmm, mit keres itt ez az üres naplóbejegyzés? Beszélni kellene Jim fejével (és esetleg összehasonlítani ezt az előző változattal az `rcsdiff` segítségével). De honnan tudjuk, melyik naplóbejegyzés melyik változathoz tartozik? Használjuk a `-v` kapcsolót az `rcs2log` parancshoz:

```
root@www:/usr/local/apache/htdocs#
➥ rcs2log -v index.html
2002-08-14 rob <rob@mouse>

* index.html 1.54: meeting announcement

2002-07-30 rob <rob@mouse>

* index.html 1.53: gre.tunnel announcement

2002-07-12 sderle <sderle@mouse>

* index.html 1.52:
➥ added Marin and shuffled around a bit.
```

```
2002-07-10 rob <rob@mouse>

* index.html 1.51: meeting announcement + v0.81

2002-07-01 jim <jim@mouse>

* index.html 1.50: *** empty log message ***

2002-06-20 rob <rob@mouse>

* index.html 1.49: meeting reminder
```

Most már összehasonlíthatjuk bármely két változatot:

```
root@www:/usr/local/apache/htdocs#
➥ rcsdiff -r1.49 -r1.50 index.html
===================================================
================
RCS file: RCS/index.html,v
retrieving revision 1.49
retrieving revision 1.50
diff -r1.49 -r1.50
199a200,202
> <dt><a href=
➥ "http://labs.google.com/">Google Labs</a></dt>
> <dd>Take a look at the strange
➥ and wondrous things that Google is up to...</dd>
>
```

Tehát csupán egy hivatkozás került az oldalra. Ennek ellenére, bármilyen kényelmes is a ^D megnyomásával elintézni a naplózást, általában véve hibásnak tekinthető, ha *semmilyen* bejegyzést nem készítünk.

Mindenesetre néhány egyszerű parancs, a `ci`, a `co`, az `rcsdiff` és az `rcs2log` elsajátításával nagyon rugalmas változatkövető rendszer birtokába juthatunk. Használjuk ezeket jól, és valószínűleg egy napon az életünket köszönhetjük nekik. Jól van na, de legalábbis a második legdrágábbat: az adatainkat.

# #26 A CVS alapjai

## A Concurrent Versioning System ábécéje

A Concurrent Versioning System (CVS) állományok egy idejű fejlesztésére szolgáló rendszer. Általánosan használják nagy programfejlesztési munkákhoz, de jól hasznosíthatják a rendszergazdák is, továbbá bármilyen műszaki leírások készítői is, azaz mindenki, akinek állományokat kell karbantartania. (A CVS-ről színvonalas magyar nyelvű cikk jelent meg a Linuxvilág hasábjain, lásd www.linuxvilag.hu.)

A CVS egy központi lerakóban (*repository*) tárolja a fájlokat. Ehhez a fájlok minden felhasználója számára hozzáférés biztosítható, szabályos Unix-hozzáférési engedélyek segítségével. A parancsok lehetővé teszik egy fájl kikérését a fejlesztéshez, és a változtatások véglegesítését a központi lerakóban. A CVS ellenőrzi a fájlokat a ki- és bemásolás során, hogy egyik felhasználó munkája se írja felül másokét.

A rendszer lehetővé teszi a fájl történetének megőrzését, ami rendkívül hasznos, ha a főnök úgy dönt, hogy mégis az a szolgáltatás kellene a programba, amit hónapokkal korábban már töröltünk. Emellett biztosítja, hogy a lerakó mentése az egész munkafolyamat mentésére elegendő legyen (feltéve, hogy minden szükséges állomány a lerakóban van).

### Jellemző alkalmazási területek

A CVS programfejlesztőknek szánt rendszer, dolgozzanak akár egyedül, akár csapatban. A magányos farkasok számára a CVS olyan lerakót tesz elérhetővé, amit akár otthonról, az irodából vagy egy ügyfélgépről használhatnak anélkül, hogy lemezeket kellene hurcolászniuk. Ezenkívül változatkövetést is nyújt számukra, lehetővé téve az adatvesztés nélküli visszatérést a korábbi változatokhoz. A programozó csapatoknak nyilvántartja azt is, hogy ki melyik sort változtatta meg a kódban, és megakadályozza mások munkájának közvetlen felülírását.

A rendszergazdák a CVS-ben tarthatják a beállításfájlokat. Egy módosítást a `cvs commit` paranccsal véglegesíthetünk, azután rögvest ki is próbálhatjuk. Amennyiben nem működik, visszaállíthatjuk a változtatás előtti állapotot, még akkor is, ha a hibát csak hat hónappal később vesszük észre.

A kiszolgálótelepek beállításai is tárolhatók a CVS-ben. Új kiszolgálót kell beállítani? Elegendő a CVS-ből kikérni az adott kiszolgálótípushoz tartozó beállításokat. Az összes változtatás berakása egyben segít nyomon követni azt is, ki, mikor, mit csinált.

Ebben a részben megnézzük, mire van szükség a CVS gyors használatbavételéhez mind a kiszolgáló, mind az ügyféloldalon.

### Új lerakó létrehozása

A lerakót (*repository*) olyan gépen kell létrehozni, amelyen elegendő hely van az összes fájl és a tervezett módosítások tárolására. Egyszerű szabályként azt mondhatjuk, hogy a lerakót helyezzük olyan lemezrészre, amin a modul várható végleges méretének háromszorosa a szabad hely. Ezután alkalmazzuk a „Scotty-elvet" és szorozzuk meg a becsült számot kettővel – ugyanis a program növekedni fog. Ha bináris fájlokat akarunk tárolni, szorozzunk tízzel. Az első önálló munkánk után már érezni fogjuk, mennyi helyet kell hagynunk.

Először győződjünk meg róla, hogy minden CVS-felhasználó rendelkezik-e érvényes felhasználói névvel, és minden használni kívánt gépről hozzá tudnak-e férni a lerakóhoz.

Ezután hozzuk létre a lerakó fő könyvtárát. A lerakó leggyakrabban a */home/cvsroot* vagy a */usr/local/cvsroot* könyvtárba kerül.

```
cvs -d /home/cvsroot init
```

A Debian Linuxban létezik egy parancsfájl, a *cvs-makerepos*, ami előre megadott Debian beállításfájlok alapján létrehoz egy lerakót. A `man cvs-makerepos` paranccsal többet is megtudhatunk róla, míg a `man cvsconfig` paranccsal arról a rendszerről, amellyel Debian CVS-lerakók beállítását lehet elvégezni.

Általában a legtöbb CVS-kiszolgáló egyetlen lerakót használ, amely több modult tartalmaz. Például lehet a lerakónk a */home/cvsroot* alatt, de tartalmazhat több projektet is (például *tools, email, dns, myproj* stb.).

## Új modul importálása

Mielőtt betöltenénk a fájlokat a CVS-be, gondoljuk végig a könyvtárszerkezetet. A fájlok vagy könyvtárak mozgatása vagy átnevezése során megsérülhet a fájlok változástörténete. Egy könyvtár törlése az összes benne lévő fájl és alkönyvtár adatainkat elvesztésébe kerülhet.

Hozzuk létre a kiinduló könyvtárszerkezetet, még akkor is, ha az csak egyetlen könyvtár. Adjuk hozzá a kívánt kiindulófájlokat. A projekt fő könyvtárában adjuk ki a `cvs -d repository import modulneve vendor tag release tag` parancsot.

A legtöbb esetben nem kell többet tudnunk a `vendor` és `release` tagokról. A CVS megköveteli, hogy megadjuk őket, de használhatjuk egyszerűen a modul nevét a `vendor` tag, a változatszámot pedig a `release` tag helyett.

```
/home/jenn$ cd example
/home/jenn/example$ cvs -d /home/cvsroot import
➥ example example_project ver_0-1
```

## Környezeti változók

Ha csak egyetlen lerakóval dolgozunk és meg szeretnénk takarítani némi gépelést, akkor a következőképpen állítsuk be a `$CVSROOT` környezeti változóban a lerakó teljes elérési útvonalát:

```
export CVSROOT=/home/cvsroot/
```

Ha a `$CVSROOT` értékét megadtuk, elhagyható a `-d repository` rész az összes CVS parancsból. Ha időnként másik lerakóval is dolgozunk, vagy a `$CVSROOT` változót állítsuk át, vagy használjuk a `-d` kapcsolót a változó felülbírálására.

Ha a CVS-lerakó másik gépen található, meg kell mondanunk a CVS-nek hogyan férhet hozzá. A távoli lerakók estében a `$CVSROOT` változót `:elérésimód:felhasználó@gép:` útvonal formában kell megadni. Valahogy így kell festenie:

```
:ext:jenn@cvs.example.com.au:/home/cvs.
```

Alapértelmezés szerint a CVS az `rsh` segítségével fér hozzá a távoli lerakóhoz. Bár ez az átviteli módszer ésszerű választásnak tűnhetett néhány évvel ezelőtt, az `rsh` futtatása valódi munkára használt kiszolgálón *nagyon rossz ötlet.* Szerencsére az `ext` hozzáférési mód azt adja meg, hogy a CVS a `$CVS_RSH` változót nézze meg, és a benne megbúvó program futtatásával kapcsolódjon a lerakót tartalmazó géphez. A legtöbben a `$CVS_RSH` változóba az `ssh` értéket rakják és az `ssh` ügyféloldali kulcsait állítják be (lásd [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal) a kiszolgálón. Ez nagymértékben egyszerűsíti a munkát, mert nem kell minden egyes alkalommal jelszót megadni, ha véglegesítünk egy változtatást.

A másik gyakran használt hozzáférési mód a *pserver*, amit manapság leginkább nyilvános, anonim CVS-lerakók eléréséhez használunk. Az anonim CVS használatáról (és a saját nyilvános lerakó beállításáról lásd a [#36] „Biztonsági mentés a *pax* programmal" részt).

### Kapcsolódó anyagok

- CVS in a Nutshell (O'Reilly)
- [#36] CVS: anonim lerakók
- [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal

## #27 CVS: modulok kikérése

### Hogyan szerezzünk be munkapéldányt egy CVS-modulból?

A CVS központi lerakóban tárolja a modulokat, de a felhasználók a saját munkapéldányukon dolgoznak.

Hozzunk létre egy könyvtárat a másolat számára (én általában a *~/cvs* könyvtárat használom), majd lépjünk is be a könyvtárba.
Az `example` nevű modul kikérésére használjuk a `cvs checkout example` parancsot.

A kikérés eredményeként a modul állományai és alkönyvtárai a *cvs* könyvtárba másolódnak.

```
cvs$ ls
example
cvs$ cd example; ls
CVS src
cvs/example$ cd CVS; ls
Entries Repository Root
```

A *cvs* olyan különleges könyvtár, amit a CVS saját célra használ. A *CVS/Entries* azokat a fájlokat és alkönyvtárakat sorolja föl, amelyekről a CVS tud. A *CVS/Repository* tartalmazza a megfelelő könyvtár teljes hozzáférési útvonalát a lerakóban. A *CVS/Root* tartalmazza a lerakó elérési útvonalát, tehát ezekhez az állományokhoz nem kell többé a `-d` kapcsolót használni.

Jegyezzük meg azonban, hogy a *CVS/Root* felülbírálja a `$CVSROOT` környezeti változót, tehát ha lerakót változtatunk, kérjük ki újra a modult, vagy ha éppen egy nagyszabású módosítás közepén vesszük észre, hogy a lerakót meg kell változtatni, próbáljuk ki a következő Perl egysorost (lásd [#73] Keresés és helyettesítés Perllel):

```
cvs/src$ ls
CVS Makefile sample.h sample.c
```

A példában szereplő forrásfájlok az *scr* könyvtárban helyezkednek el. A *sample.c*, a *sample.h*, valamint a `Makefile` egyszerű állományok a munkapéldányban. A lerakóban a módosítások nyilvántartó formában tárolódnak.

## #28 CVS: a munkapéldány frissítése

### A CVS-modul legutóbbi változásainak letöltése

Az összes olyan alkalommal, amikor valaki változtatásokat véglegesített, illetve minden nap a munka megkezdése előtt lépjünk be a munkakönyvtárba és adjuk ki a `cvs update` parancsot. Ez összehasonlítja a munkapéldányt a lerakó állományaival és az összes változtatást letölti. A `cvs update -d` az estleges új könyvtárakat is letölti.

Az `update` minden fájl állapotát megjeleníti ellenőrzés közben:

- `U` – az állomány sikeresen frissítve.
- `A` – az állomány hozzáadva, nem véglegesítve (`cvs commit` parancs szükséges).
- `R` – állomány törölve, nem véglegesítve (`cvs commit` parancs szükséges).
- `M` – állomány módosítva a munkakönyvtárban; a lerakóban lévő fájl megváltozott és a munkapéldány régebbi volt, mint amikor a CVS legutóbb ellenőrizte, vagy a lerakó olyan változtatásokat tartalmazott, amiket a rendszer biztonságosan be tudott illeszteni.
- `C` – ütközés a lerakóban lévő fájl és a munkapéldány között, emberi beavatkozás szükséges.
- `?` – a fájl létezik a munkakönyvtárban, de a lerakóban nincs, a CVS nem tud vele mit kezdeni.

# #29 CVS: a tagok használata

## A modulváltozatok közvetett jelölése a CVS-ben

A tagok közvetett (symbol) nevet rendelnek egy fájlváltozathoz vagy fájlok csoportjához. A `cvs tag` parancs a jelenlegi munkakönyvtár és alkönyvtárai összes állományának lerakóban lévő változatához nevet rendel.

A `csv tag` parancs egy időbélyeget vesz alapul, nem a munkakönyvtár állományainak módosítási idejét nézi. Közvetett vagy szimbolikus nevet rendel minden fájl és alkönyvtár időbélyegnél régebbi, hozzá legközelebb eső változatához.

Vegyük figyelembe, hogy a CVS nem engedi meg a . (pont) használatát a tagokban. A fájl nevét a tagnév után adjuk meg, mint a következő példában is tettük:

```
cvs tag tagnév fájlnév
cvs tag tagnév
```

Egyébként a parancs a jelenlegi munkakönyvtár minden állományának lerakóbeli változatához nevet rendel:

```
cvs tag -c tagnév
```

A `-c` kapcsoló hatására a folyamat leáll, ha a lerakóban tárolt változat a munkapéldánytól különbözik:

```
cvs/example$ cvs tag release-1-0 src/sample.c
cvs/example/src$ cvs tag release-1-0
cvs/example/src$ cvs tag -c release-1-0
```

A jelölt változatok letöltéséhez használjuk a `checkout` vagy az `update -r` kapcsolóját. Ha a szokásos munkakönyvtárban futtatjuk a `checkout` vagy az `update` parancsot, a megnevezett változat felülírja a meglévő fájlokat, mint a következő példában is láthatjuk:

```
cvs checkout -r tagnév
cvs update -r tagnév
```

Következzen egy példa:

```
cvs$ mkdir example-rel-1.0
cvs/example-rel-1.0$ cvs checkout -r release-1-0
```

vagy

```
cvs/example$ cvs update -r release-1-0
```

## #30 CVS: a modulok megváltoztatása

### A módosítások véglegesítése a CVS-ben

Ha lekértük a fájlokat, a szokásos módon szerkeszthetjük és fordíthatjuk a programkódot. A változtatásokat a `cvs commit` paranccsal véglegesítjük a lerakóban. Ezt a parancsot a könyvtárfában mindig magasabb helyen kell kiadni, mint ahol a módosított fájlok vannak – a munkapéldány gyökérkönyvtárában mindig kiadhatjuk.

Használhatjuk a `cvs commit` *`fájlnév`* parancsot is, ezzel egy fájlt vagy egy könyvtár és alkönyvtárai tartalmát véglegesíthetjük.

A különböző fejlesztőcsapatok véleménye eltér arról, hogy milyen gyakran kell véglegesíteni a változtatásokat. A jól bevált szabályok közé tartozik a „minden sikeres fordítás után" meg az „ebéd előtt, és mielőtt hazamegyek".

```
cvs/example$ cvs commit
cvs commit: Examining .
cvs commit: Examining src
jenn@cvs.sample.com.au's password:
```

A CVS megvizsgál minden könyvtárat a jelenlegi munkakönyvtár alatt. Minden ismert fájlban ellenőrzi a változásokat. Ha a CVS-lerakó távoli gépen van, a CVS jelszót fog kérni, hacsak nem állítottuk be az SSH-ügyfélkulcsokat (lásd [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal), amivel elkerülhető a jelszavas azonosítás.

A CVS ezután megnyitja a rendszer alapértelmezett szerkesztőjét – a `$CVSEDITOR` vagy a `$EDITOR` környezeti változók alapján. Írjuk be a megfelelő fájlokhoz tartozó módosítási megjegyzéseket, például:

```
CVS:----------------------------------------------
------------------
CVS: Enter Log. Lines beginning with 'CVS:'
➥ are removed automatically
CVS:
CVS: Committing in .
CVS:
CVS: Modified Files:
CVS: example/src/sample.h example/src/sample.c
CVS:----------------------------------------------
------------------
```

Nagyon ajánlatos értelmes bejegyzéseket készíteni, ugyanis ha egy korábbi változathoz vissza akarunk térni és a megjegyzésben csak ennyi áll: „egy pár hiba kijavítva", akkor a `cvs diff` parancs nélkül nem fogjuk tudni, hogy melyik változathoz kell visszatérnünk.

Amennyiben ütközés lehetőségét észleli, a `cvs commit` nem fut le. Ezt helyesbítsük a `cvs update` használatával – a CVS megpróbálja összeilleszteni a fájlokat, és emberi segítséget kér, ha ez adatvesztés nélkül nem lehetséges:

```
cvs server:
➥ Up-to-date check failed for 'cvs_intro.html'
cvs [server aborted]: correct above errors first!
cvs commit: saving log message in /tmp/cvst7onmJ
```

## #31 CVS: fájlok egyesítése

### A frissítési ütközések feloldása a CVS-ben

Ha a CVS nem képes egyesíteni egy módosított állományt a lerakóban lévő változattal, jelenti az ütközést a `cvs update` parancs kimenetében. Az eredeti fájlt egy *#fájl.változat* nevű állományba menti a fájl munkakönyvtárában, és az egyesítés eredményét az eredeti fájlban tárolja:

```
cvs/example$ cvs update
jenn@cvs.example.com.au's password:
cvs server: Updating .
RCS file: /home/cvs/example/sample.c,v
retrieving revision 1.3
retrieving revision 1.4
Merging differences between 1.3 and 1.4
➥ into sample.c
rcsmerge: warning: conflicts during merge
cvs server: conflicts found in sample.c
C sample.c
```

A CVS az ütköző sorokat CVS-tagok közé foglalva az egyesített fájlba írja. A CVS nem tudja önműködően feloldani az olyan ütközéseket, amikor két állományban ugyanaz a sor változott:

```
<<<<<<< sample.c
Szándékosan létrehozott ütközés.
=======
Ütközzünk!
>>>>>>> 1.4
```

# #32 CVS: fájlok és könyvtárak hozzáadása és eltávolítása

## Fájlok hozzáadása és törlése a modulban

A véglegesítési folyamat és a `cvs update` után azokat a fájlokat, amikkel a CVS nem tud mit kezdeni kérdőjellel megjelöli. Ezeket hozzá kell adni a lerakóhoz, hogy a CVS fel tudja ismerni őket.

Használjuk a `cvs add` *`fájlnév`* parancsot, amivel az új fájlokat hozzáadásra jelöljük ki. Ezután a CVS a legközelebbi véglegesítéskor hozzáadja a lerakóhoz a fájlokat.

### Fájlok eltávolítása

A `cvs remove` *`fájlnév`* paranccsal távolíthatunk el fájlokat a munkapéldányból. Ahhoz, hogy a CVS a lerakóból eltávolítsa a fájlt, le kell törölnünk a fájlrendszeren. Tulajdonképpen akkor sem távolítja el egészen, hanem a lerakó *Attic* nevű könyvtárába helyezi.

### Könyvtárak eltávolítása

CVS-parancsokkal nem lehet könyvtárakat eltávolítani a lerakóból. Ha egy könyvtárra már nincs szükség, ürítsük ki a `cvs remove` paranccsal, és használjuk a `-P` kapcsolót, amikor letöltjük a munkapéldányt a `cvs update` és `cvs checkout` parancsokkal. A `-P` kapcsoló biztosítja, hogy üres könyvtárak ne töltődjenek le.

Ha nagyon muszáj, a könyvtárat az `rmdir` paranccsal eltávolíthatjuk a lerakóból. Először egy másolaton próbáljuk ki, hogy nem teszünk-e tönkre valamit: ha a könyvtárban a lerakón lévő változatban van *Attic* könyvtár, akkor el fognak veszni a benne tárolt adatok.

Ha egy könyvtárat eltávolítunk, akkor ugyanezt az összes felhasználónak is el kell távolítania, és ki kell kérniük a modul új változatát.

# #33 CVS: a fejlesztés elágaztatása

## Új fejlesztési ág létrehozása a CVS-ben

Ha a kód egy régebbi változatában ki kell javítani egy hibát anélkül, hogy a jelenlegi kódot megváltoztatnánk, vagy módosított beállításokat akarunk létrehozni a próbakiszolgálóink számára anélkül, hogy

a működő kiszolgálók beállításait megváltoztatnánk, szükségessé válhat a modulok elágaztatása. Az új ág lehetővé teszi a főmodul egy változatának tárolását és nyomon követését anélkül, hogy hatással lenne a főmodulra. Az ágon végzett változtatásokat később be lehet illeszteni. Az ág létrehozásához használjuk a `cvs tag -b ágtag` parancsot, ahogyan a következő példában is láthatjuk:

```
cvs/example$ cvs tag -b release-1-0-patches
```

Az ágat a `checkout` vagy az `update` paranccsal tölthetjük le.
A `checkout` új könyvtárat hoz létre az új ágnak, az `update` pedig felülírja a munkakönyvtárat.

```
cvs checkout -r ágtag
cvs update -r ágtag
```

Például:

```
cvs/example-rel-1.0$
➥ cvs checkout -r release-1-0-patches
cvs/example$ cvs update -r release-1-0-patches
```

Az ágakat vissza lehet illeszteni a főágba, a `cvs update` és a `cvs commit` által használt ütközésfeloldó rendszer segítségével.

```
cvs checkout modul
cvs update -j ágtag
```

Például:

```
/tmp/example$ cvs checkout example
/tmp/example$ cvs update -j release-1-0-patches
```

Vagy egy parancsban:

```
cvs checkout -j ágtag modul
```

A következő példa a megtalált ütközéseket feloldja:

```
/tmp/example$
➥ cvs checkout -j release-1-0-patches example
```

Ezután adjuk ki a `cvs commit` parancsot.

# #34 CVS: fájlok figyelése és lezárása

## Figyelmeztetés beállítása elektronikus levélben CVS fájlokhoz

Számos más változatkövető rendszertől eltérően a CVS nem ad lehetőséget a fájlok lezárására – nem akadályozza meg a fájlok egyidejű változtatását. Viszont az beállítható, hogy a rendszer egyes fájlokat figyelemmel kísérjen, és a figyelést igénybe vevő felhasználóknak levelet küldjön, ha a fájlt szerkesztik. Ha egy fájlt figyelnek, akkor a fejlesztőknek a `cvs edit` és `cvs unedit` parancsokkal kell a fájlt szerkesztésre kiadni. A megfigyelés nélküli fájlok módosíthatók a CVS bármiféle értesítése nélkül.

A fájlokat a következőképpen jelölhetjük ki figyelésre:

```
cvs watch on (fájlok)
cvs watch off (fájlok)
```

A figyelést a következő módon vehetjük igénybe:

```
cvs watch add (fájlok)
cvs watch remove (fájlok)
```

vagy

```
cvs watch add -a edit|unedit|commit|all (fájlok)
cvs watch remove -a edit|unedit|commit|all (fájlok)
```

A *notify* nevű CVS fájl adja meg, hogy mi történjen, ha egy figyelt állomány megváltozik. Az alapbeállítás szerint levelet küld a felhaszná-

lónak a CVS-kiszolgálóra. Ha a felhasználók más címeket használnak, állítsuk be a *users* fájlt a *CVSROOT* könyvtárban. A bejegyzések `felhasználó:cím` alakúak, külön sorban.

```
jenn:
jenn@cvs.example.com.au
```

## #35 CVS: a CVS biztonsága

### Védjük a felhasználókat és a CVS-ben tárolt kódot!

#### Távoli lerakók

Ha a lerakó a helyi gépen van, a biztonság és a hozzáférés kérdése is elég egyszerűen kezelhető. Megadható a CVS-lerakó fő könyvtára a `$CVSROOT` környezeti változóban, vagy a `checkout` parancs `-d` *`könyvtár`* kapcsolójával.

Ha a lerakó távoli gépen helyezkedik el, meg kell mondanunk a CVS-nek, hogy melyik gépen találja, és milyen módszert használjon a hozzáféréshez. Több módszer közül lehet választani, de biztonsága és egyszerűsége folytán én az SSH híve vagyok. A távoli `$CVSROOT` megadásának formája:

```
:mód:[[felhasználó]:[jelszó]@]gép[:[kapu]]
➥ :/hozzáférési/útvonal
```

Például a

```
:ext:jenn@cvs.example.com.au:/usr/local/cvsroot
```

(Megjegyzem, az `info cvs` nem egészen azt írja, amit a gépemen lévő CVS elfogad. Azt a formát adtam meg, ami nekem működött – kettősponttal a gépnév és a hozzáférési út között. Mindenki használja a gépén jól működő változatot.)

Az SSH használatát az `ext` beállítással adjuk meg. Ez az üzemmód külső `rsh` vagy `rsh`-megfelelő programot használ a CVS-kiszolgáló eléréséhez. Az `rsh` helyett olyan módon használhatunk SSH-t, hogy

a $CVS_RSH változóban a ssh értéket adjuk meg. Győződjünk meg róla, hogy az SSH a kiszolgálón és az ügyfélgépen is fut, és az SSH-kulcsok létrehozása megtörtént, valamint a felhasználóknak mindkét gépen van felhasználói neve és jelszava. Ha a felhasználói név ugyanaz, akkor a *felhasználó@* rész elhagyható a CVSROOT változóból. Ha szabályos SSH-kaput használunk, a kaput nem szükséges megadni.

```
cvs -d :ext:cvs.example.com.au:/usr/local/cvsroot
➥ checkout sample
```

### Hozzáférési jogosultságok

A lerakó minden állománya csak olvasható. Ezeknek a fájloknak az engedélyeit ne változtassuk meg. A hozzáférési jogosultságokat a könyvtárak engedélyeivel szabályozzuk. A legtöbb rendszergazda felhasználói csoportot hoz létre azokból a felhasználókból, akiknek hozzá kell férniük az adott modulhoz, és a csoportnak írási engedélyt is ad a könyvtárra.

Ha távoli lerakót használunk, a modul gyökérkönyvtárára *setgid* beállítást használjunk, hogy az összes alkönyvtár megfelelő engedélyekkel jöjjön létre. Helyi lerakó esetében a $CVSUMASK változóval szabályozhatók a lerakó állományainak és könyvtárainak hozzáférési jogosultságai.

### Fejlesztői gépek

A kód biztonságáról akkor gondoskodhatunk, ha a lerakót és az összes kikért példányt biztosítjuk – ez általában a fejlesztők gépeit jelenti. Nem elég tudni, hogy a lerakó biztonságos és minden adatátvitel titkosítottan zajlik, ha valaki egyszerűen besétálhat az egyik fejlesztő szabadnapján az irodájába és CD-re írhatja az egész anyagot. Tartsuk be a hálózati és fizikai biztonság szokásos szabályait a fejlesztői gépeken is, a próba- és bemutatóváltozatok esetében, illetve minden egyéb helyen, ahová a kódot kikérik.

# #36 CVS: anonim lerakók

## Hozzunk létre saját csak olvasható anonim CVS-lerakót

### Anonim lerakó létrehozása

A *pserver* mód lehetővé teszi, hogy egy távoli lerakót elérjünk a felhasználói név és jelszó megadásával; a jelszófájlt kezelheti a CVS, vagy használható a rendszer */etc/passwd* fájlja. A *pserver* sajnos titkosítás nélkül küldi el az adatokat, tehát egy hallgatózó legjobb esetben is megszerezheti a CVS bejelentkezéshez használt adatokat, vagy legrosszabb esetben hozzáférhet az egész géphez. Ezért a legtöbben a távoli lerakót SSH használatával érik el. Ezzel kapcsolatban a további részleteket lásd a [#35] „Védjük a felhasználókat és a CVS-ben tárolt kódot" trükkben.

Természetesen felesleges (és kényelmetlen) lenne SSH-t használni akkor, ha a forráskódunkhoz csak olvasható, nyilvános hozzáférést biztosítunk mindenkinek. Itt mutatkozik meg a *pserver* haszna: könnyen adhatunk vele anonim hozzáférést.

Mielőtt működésbe hozzuk az anonim CVS-t, be kell állítanunk a kiszolgálót a hagyományos *pserver* eléréséhez.

### A pserver beállítása

Mivel a *pserver* anonim CVS-elérést fog adni, létre kell hoznunk egy felhasználót, akinek semmilyen írási joga nincs a lerakóra. Hozzunk létre egy anonymous nevű felhasználót (akit zavarnak a 9 betűs felhasználói nevek, általában a cvsanon nevet használja). Állítsuk be a */bin/true* héjat, saját könyvtárnak pedig valami ártalmatlan helyet (például a */var/empty*-t), hozzunk létre neki saját csoportot, és zárjuk le a jelszavát (erre a `passwd -l` kellően gyors módszer). Ez a felhasználó soha nem fog belépni a rendszerre, csak a CVS számára hoztuk létre.

Ezután létrehozunk egy CVS jelszófájlt. Írjuk a következő sort a *CVSROOT/passwd* nevű fájlba a lerakó könyvtárában:

```
anonymous:23MLN3ne5kvBM
```

Ha cvsanon nevű felhasználót hoztunk létre, a következőt használjuk:

```
anonymous:23MLN3ne5kvBM:cvsanon
```

A CVS *passwd* állományában a bal oldali bejegyzés a CVS felhasználói név, ezt követi a titkosított jelszó, végül egy esetleges rendszerbeli felhasználónév következik, amihez a CVS-név hozzárendelhető. Ha ezt nem adjuk meg, a CVS a rendszer *passwd* állományából keresi ki az első bejegyzést, és azt használja. A kódolt szöveg az anonymous szó.

Annak érdekében, hogy az anonymous felhasználó semmi esetre se változtathassa meg a lerakó tartalmát, írjuk az `anonymous` bejegyzést a *CVSROOT/readers* fájlba a lerakó könyvtárába. Ez a fájl a benne (külön sorban) felsorolt felhasználókat csak olvasási jogosultsággal rendelkezőként jelöli meg.

Ezután be kell állítanunk, hogy a CVS soha ne fogadjon el *pserver*-belépést más felhasználói névvel (hogy önfejű felhasználóink nehogy ötletszerűen a rendszerbeli jelszavukat használják a *pserver*-re belépéshez). Ezt a lerakó könyvtárában lévő *CVSROOT/config* állomány szabályozza. Vegyük ki a megjegyzésjelet a `SystemAuth=no` sor elől, így csak a *CVSROOT/passwd* fájlban megadott felhasználók léphetnek be a *pserver* használatával. Ez nem érinti az `ext-` és `ssh`-hozzáférést használó felhasználókat; az ő hozzáférésüket továbbra is egyszerű fájlrendszer-jogosultságok szabályozzák, nem a CVS *passwd* állománya.

Végül megmondhatjuk a rendszernek, hogy most már fogadhat *pserver* kapcsolatokat. A CVS nem is futtatható démonként; az `inetd` indítja el. A 2401 kaput használja, tehát a következő sort adjuk a */etc/services* fájlhoz:

```
pserver 2401/tcp
```

A következő bejegyzést pedig a */etc/inetd.conf* állományhoz:

```
pserver stream tcp nowait root /usr/bin/cvs cvs
➥ --allow-root=/usr/local/cvsroot pserver
```

A */usr/loval/cvsroot* helyére írjuk a lerakó könyvtárát. Most már kiadhatjuk az `skill -HUP inetd` parancsot (ha nincs a gépen a `skill` telepítve, akkor lásd [#18] „Rendszererőforrások felügyelete az egyes folyamatok szintjén" részt), és működik is.

### Távoli pserver használata

Az újonnan létrehozott anonim lerakó kipróbálásához először állítsuk be a következőt a `$CVSROOT` környezeti változóban:

```
:pserver:anonymous@your.machine.here
➥ :/usr/local/cvsroot
```

Mielőtt a `cvs checkout` parancsot kiadhatnánk, be kell lépnünk a `cvs login` paranccsal. Ezután jöhet a `cvs checkout` modul, és a szokásos letöltést kell látnunk. Ellenőrizzük, hogy nem tudunk-e a `cvs checkin` paranccsal írni a lerakóba, és már ki is léphetünk az anonim CVS-sel a nagyvilág színe elé.

# Biztonsági mentések

## 37–44. trükkök

Ne legyenek kétségeink, mert egy napon megtörténik az elképzelhetetlen... A merevlemezek kopnak, a szalagok nyúlnak és szakadnak. Az is megeshet, hogy az ember véletlenül rossz kapcsolót ad meg egy parancshoz (vagy „helytelen" könyvtárban, esetleg nem a megfelelő gépen írja be az utasítást), és azon kapja magát, miközben adatai köddé válnak, csúf szavakkal szidja saját magát az ENTER billentyű leütése miatt. Az ilyen pillanatokban tanuljuk meg a jól megtervezett és kivitelezett biztonsági mentések igazi értékét.

Bár megadhatjuk a kezdő lökést az adatmentés néhány érdekes megközelítésének bemutatásával, ez a fejezet semmiképpen sem öleli fel az egész témát. Rendszergazdaként ki kell dolgoznunk az adott helynek megfelelő tervet a biztonsági mentések végrehajtására, a tervet meg kell valósítanunk és igény szerint folyamatosan fel kell újítanunk. Semmiképpen ne feltételezzük, hogy csak azért, mert beállítottunk egy `cron` feladatot a mentés elvégzésére, maguk a másolatok jók (vagy egyáltalán maga a `cron` megfelelően működik). Folyamatosan kísérjük figyelemmel a naplókat, ne csak ötletszerűen ellenőrizzük. Vegyünk új tárolóeszközöket, mielőtt a régi betegeskedni kezdene. A másolatok ellenőrzését tegyük rendszeres elfoglaltságunkká és

végezzünk belőlük teljes helyreállítást olyan gyakorisággal, amilyen gyakran szükségét érezzük ahhoz, hogy nyugodtan tudjuk álomra hajtani fejünket. Még így is fontoljuk meg, hogy ne tartsunk-e még egy biztonsági másolatot a legfontosabb adatainkról, lehetőleg nem a helyszínen. Rendkívül részletesen tárgyalja a megbízható biztonsági mentési terv készítésének minden szempontját W. Curtis Preston könyve, a *Unix Backups & Recovery* (O'Reilly).

Ebben a részben áttekintünk néhány módszert arra, hogy amikor eljön az igazság pillanata, a biztonsági másolataink mindig kéznél legyenek. Minden kiszolgáló más, és minden feladatnak megvan a maga igényszintje a helyreállítási szolgáltatások terén (a heti növekményes mentéstől az öt perces pillanatfelvételekig). Nem próbáljuk meghatározni a tökéletes mentési stratégiát, inkább olyan eszközöket ismertetünk, melyeket mindenki használhat, amikor kialakítja a sajátját.

## #37 Biztonsági mentés SSH-kapcsolaton keresztül a tar programmal

### Másoljuk a fájlrendszer tetszőleges bitjeit a kiszolgálók között SSH és tar használatával

Az `scp` segítségével könnyen másolhatunk fájlokat a kiszolgálók között:

```
root@inky:~# scp some-archive.tgz blinky:/
```

Akár több fájlt is másolhatunk egyszerre:

```
root@pinky:~/tmp# scp clyde:/usr/local/etc/* .
```

De az `scp`-t nem arra tervezték, hogy alkönyvtárakat másoljon, vagy hozzáférési jogosultságokat őrizzen meg. Az SSH program készítése során az egyik legkorábbi (és szerintem legragyogóbb) tervezési döntés az volt, hogy pontosan úgy működjön, mint minden más szabályos Unix-parancs. Amikor felhasználói beavatkozás és belépés nélkül parancsok végrehajtására használjuk, az SSH egyszerűen a szabályos bemenetről olvas, és a szabályos kimenetre írja az eredményt. Minden egyes csővezetékre, amiben az SSH szerepel, gondoljunk

olyan módon, mint egy másik gépre nyíló, egyszerűen létrehozható kapura. Tegyük fel, hogy az egyik kiszolgáló */home* könyvtárát át akarjuk másolni egy másikra:

```
root@inky~# tar zcvf - /home |
➥ ssh pinky "cat > inky-homes.tgz"
```

Vagy akár a tömörített másolatot írjuk közvetlenül a másik gép szalagos meghajtójára:

```
root@blinky~# tar zcvf - /var/named/data |
➥ ssh clyde "cat > /dev/tape"
```

Tegyük fel, hogy egy könyvtárat át szeretnénk másolni az egyik gépről közvetlenül a másik fájlrendszerébe. Ebben a példában Apache működik a helyi gépen, de a távolin nem. Hangoljuk össze a kettőt:

```
root@clyde:~# cd /usr/local
root@clyde:/usr/local# tar zcf - apache/ \
   | ssh pacman "cd /usr/local;
     ➥ mv apache apache.bak; tar zpxvf -"
```

Ezzel a pacman gépen a */usr/local/apache* könyvtárat mozgatja a */usr/local/apache.bak/* alá, azután létrehozza a clyde gép */usr/local/apache* könyvtárának pontos másolatát, megőrizve a hozzáférési jogosultságokat és a teljes könyvtárszerkezetet. Kísérletezhetünk tömörítés alkalmazásával mindkét oldalon (a `tar z` kapcsolójával). A teljesítmény függ mind a két gép feldolgozási sebességétől, a hálózat sebességétől (és terhelésétől), továbbá attól is, hogy használtunk-e már tömörítést az SSH-ban.

Végül tételezzük fel, hogy a helyi gépen egy nagyméretű `tar`-csomag terpeszkedik, és ennek tartalmát kívánjuk helyreállítani a távoli gépen anélkül, hogy előbb odamásolnánk (mondjuk valóban óriási a csomag, és bár elegendő hely van a kicsomagolt adatoknak, de magának a csomagnak már nem marad tárterület):

```
root@blinky~# ssh pinky "cd /usr/local/pacland;
➥ tar zpvxf -" \
    < nagyon-nagy-csomag.tgz
```

Esetleg közelítsünk másik irányból:

```
root@pinky:/usr/local/pacland#
➥ ssh blinky "cat nagyon-nagy-csomag.tgz" \
  | tar zpvxf -
```

Ha hiba keletkezik a távoli gépen a kibontott vagy létrehozott csomagokban, először ellenőriznünk kell, hogy a távoli gépen a *~/.bashrc* fájlunkban nincs-e valami, ami a terminálra ír, például a */usr/games/fortune* vagy más, terminálra író program. Célszerűbb a *~/.bash_profile* vagy a *~/.bash_login* fájlból indítani, mint a *~/.bashrc* fájlból, mert legfeljebb egy bejelentkező embert érdekelhet mit ír ki a *fortune* program, de egy csővezeték részeként végrehajtott távoli parancsot semmiképpen nem érint. Továbbra is beállíthatunk környezeti változókat a *~/.bashrc* fájlban, és más programokat is futtathatunk, feltéve, ha ezek biztosan nem írnak semmit a szabványos kimenetre (STDOUT) vagy a hibacsatornára (STDERR).

Ha SSH-kulcsokat használunk nincs szükség jelszavak megadására, valamint egyszerűbb a fájlrendszer tetszőleges részeit hálózaton keresztül másolni (ráadásul szükség esetén könnyűszerrel írható hozzá `cron` feladat). Az alábbi hivatkozások segítenek bennünket, hogy minden kiszolgálóhoz gyors (és biztonságos) SSH-hozzáférést állítsunk be. (Fontos megemlíteni, hogy már létezik olyan rendszermagfolt, amelyik segítségével távoli fájlrendszereket tudunk SSH-val befűzni és helyi meghajtóként kezelni – Lektor.)

### Kapcsolódó anyagok

- [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal
- [#67] Villámgyors SSH-bejelentkezés
- [#68] SSH-Agent, hatékonyan

## #38 rsync az SSH-n keresztül

### Hangoljuk össze nagy méretű könyvtárrendszereinket az rsync paranccsal!

A `tar` SSH-n keresztül használva eszményi megoldást kínál a fájlrendszer egyes részeinek távoli gépre másolására, az *rsync* pedig a teljes fájlrendszer összehangolására ad lehetőséget a két gép között.

Általában a `tar` programmal készítjük az első másolatot, a későbbi változtatásokat pedig az *rsync* viszi át. Ez azért történik ilyen módon, mert a `tar` jellemzően akkor gyorsabb, amikor a célállományok egyike sem létezik, viszont az *rsync* sokkal fürgébb, amikor a két fájlrendszer között csak néhány apró különbség van.

Ha az *rsync* az SSH-n keresztül fut, használjuk a `-e` kapcsolót:

```
root@rover:~# rsync -ave ssh
➥ greendome:/home/ftp/pub/ /home/ftp/pub/
```

Figyeljük meg a forrásoldalon (a greendome gépen) az útvonal megadásában a záróperjelet (/). A forrás útvonalában a záróperjel (/) azt jelenti, hogy az *rsync* a könyvtár *tartalmát* másolja, de a könyvtárat magát ne. Ahhoz, hogy a könyvtár is átkerüljön, hagyjuk el a záróperjelet:

```
root@village:~# rsync -ave ssh bcnu:/home/six .
```

Ezzel a village nevű gép *~root/six/* könyvtárát hangoltuk össze a *bcnu:/home/six/* tartalmával.

Alapértelmezés szerint az *rsync* csak átmásolja a fájlokat és könyvtárakat a célba (destination copy), de nem törli őket a forrásból. A másolat pontos egyezése a `--delete` kapcsolóval érhető el:

```
six@jammer:~/public_html# rsync -ave ssh --delete
➥ greendome:~one/reports .
```

Ha a régi jelentéseket töröljük a greendome gép *~/one/reports/* könyvtárából, a jammer nevű gép *~/six/public_html/reports/* könyvtárából is törlődni fognak minden alkalommal, amikor ezt a parancsot kiadjuk. Ha egy ehhez hasonló parancsot a `cron` segítségével futtatunk, hagyjuk el a `-v` kapcsolót, mert ilyen módon megszabadulunk a fölösleges üzenetektől a kimeneten (ha az *rsync* nem hibátlanul fut, úgyis elektronikus levelet kapunk a hibaüzenettel).

Ha az *rsync* forgalmát SSH-val küldjük át, az adatok titkosítva haladnak a hálózaton, ezáltal kiaknázzuk azokat a bizalmi kapcsolatokat is, amelyeket az SSH-kulcsok használata során már kiépítettünk.
Ha két gépen nagy, összetett könyvtárrendszereket kell összehangolnunk (főleg, ha kevés köztük a különbség), az *rsync* nagyon hasznos (és gyors) segédeszköznek bizonyul.

### Kapcsolódó anyagok

- `man rsync`
- [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal
- [#68] SSH-ügynök, hatékonyan
- [#42] Önműködő pillanatfelvétel-típusú növekményes mentések az rsync programmal

## #39 Biztonsági mentés a pax programmal

### Készítsünk hordozható másolatokat – egyszerűen

A *pax*, a portable archive exchange (hordozható másolatátvitel) rövidítése, és kifejezetten a különböző Unix-változatok közötti hordozhatóság megteremtésére tervezték. Van egy kevés irónia is a névben, mivel a *pax* próbál békét teremteni a `tar` és a `cpio` között az elsőségért oly régóta folyó háborúban. A *pax* segédprogram mind a kétféle mentést képes elvégezni és a visszaállítás során felismeri a csomag típusát. Lássunk először néhány egyszerű példát a *pax* használatára, azután a cselesebb trükkökre is rátérünk.

### Másolatok létrehozása

A saját könyvtárunk tartalmának mentéséhez az írható módot a `w` kapcsolóval állítjuk be:

```
cd
pax -wf home.pax .
```

Ebben a példában a `cd` paranccsal léptem be a saját könyvtáramba, azután azt mondtam a *pax* programnak, hogy írja (`w`) egy fájlba (`f`) – melynek a *home.pax* nevet adtam – a pillanatnyi könyvtár (.) tartal-

mát. Amikor a *pax*-ot használjuk, nagyon fontos, hogy ne feledkezzünk meg az `f` kapcsoló használatáról, amivel megadjuk a létrehozni kívánt állomány nevét. Ha elfelejtjük, furcsa karakterek jelennek meg a képernyőn, borzasztó fájdalmas hangok kíséretében. Ha figyelemmel akarjuk kísérni a működését, adjuk meg a `v` (`verbose`) kapcsolót.

Nézzük meg a létrehozott fájl típusát a `file` paranccsal:

```
file home.pax
home.pax: POSIX tar archive
```

A fájl tartalmának megtekintéséhez az `f` kapcsolóval jelezzük, hogy melyik fájlt kívánjuk megtekinteni:

```
pax -f home.pax |more
```

Mivel a fájl elég hosszú, a kimenetet átirányítottam a `more` parancsra, hogy oldalanként olvashassam az archívum tartalmát.

Ha a `v` kapcsolót is megadjuk, `ls -l` stílusú alakban kapjuk meg a tartalomjegyzéket. Újfent figyeljünk rá, hogy ki ne felejtsük az `f` kapcsolót, különben semmi nem történik, csak elveszítjük a parancssorjelet (promptot) egészen addig, amíg CTRL-C-t nem nyomunk.

### Másolatok kicsomagolása

A helyreállítás (olvasás) beállításához használjuk az `r` kapcsolót, miután beléptünk a *célkönyvtárba*. Például kicsomagolom a *home.pax* nevű fájlt a saját könyvtáram *test* nevű alkönyvtárába:

```
cd test
pax -rvf ~/home.pax
```

A *pax* programmal `tar` és `cpio` állományokból is helyre lehet állítani. Képes önműködően meghatározni a megfelelő formátumot; ugyanakkor a helyreállítás előtt a *file* segédprogrammal állapítsuk meg, hogy a csomag tömörített-e. Ha igen, akkor a `z` kapcsolót kell használnunk.

Például a saját könyvtáramban van egy (~) *backup.old* nevű állomány. Először a `file` parancsot használom:

```
file backup.old
backup: gzip compressed data, deflated,
➥ last modified:
Sat Aug 17 14:21:12 2002, os: Unix
```

Mivel ez a biztonsági másolat tömörített, a következő paranccsal állítom vissza a *test* könyvtárba:

```
cd test
pax -rvzf ~/backup.old
```

A saját könyvtáramban létezik egy *backup* nevű fájl is:

```
file ~/backup
backup: cpio archive
```

Mivel ez a fájl nem tömörített, ekképpen állítom helyre:

```
pax -rvf ~/backup
```

Az, hogy az első másolat történetesen `tar`-csomag volt, a második pedig `cpio`, nem zavarta meg a *paxot*, ugyanakkor furcsa hibaüzeneteket kaptam volna, ha elfelejtem közölni a programmal, hogy az első fájl tömörített csomag.

### Felhasználói beavatkozást igénylő helyreállítás

Helyreállításkor a *pax* lehetővé tesz néhány cseles dolgot, például felhasználói beavatkozást igénylő (interactive) helyreállítást, illetve átnevezést végezhetünk az `i` kapcsoló megadásával, a következő parancs kiadásával:

```
pax -rif ~/backup
```

a *backup* nevű fájl felhasználói beavatkozást igénylő helyreállítását kezdi meg a jelenlegi munkakönyvtárba. Felhasználói beavatkozást

igénylő módban a *pax* megjeleníti minden egyes fájl nevét, és megkérdezi, hogy mit csináljunk vele: átnevezzük helyreállítás közben, helyreállítsuk az eredeti névvel, vagy ugorjuk át és ne állítsuk helyre.

```
ATTENTION: pax interactive file rename operation.
drwxr-xr-x Aug 17 15:08 .
Input new name, or a "." to keep the old name,
➥ or a "return" to skip this file.
Input >
Skipping file.
```

Ebben az esetben ENTERT nyomtam, mivel nem akartam a (.), vagyis a jelenlegi könyvtár nevét módosítani.

```
ATTENTION: pax interactive file rename operation.
drwxr-xr-x Jul 26 16:10 file1
input new name, or a "." to keep the old name,
➥ or a "return" to skip this file.
input > old
Processing continues, name changed to: old

ATTENTION: pax interactive file rename operation.
-rw-r--r-- Jun 11 00:20 file2
input new name, or a "." to keep the old name,
➥ or a "return" to skip this file.
input > .
Processing continues, name unchanged.
```

Megfigyelhető, hogy a *file1* nevét *old*-ra változtattam, a *file2* pedig változatlan maradt. A helyreállított könyvtár listázása két fájlt mutat: egy *old* és egy *file2* nevűt.

## Alkönyvtárak másolása

A *pax* egyik leghasznosabb tulajdonsága, hogy nagyon gyorsan képes lemásolni egy teljes könyvtárrendszert a merevlemez egy másik részére, másolási üzemmódban. A másolási mód használatához a következő lépések szükségesek:

1. Adjuk ki a `cd` utasítást a forráskönyvtárban.
2. Nézzük meg, hogy a célkönyvtár létezik-e; ha nem, hozzuk létre az `mkdir` paranccsal.
3. Adjuk ki a következő parancsot: `pax -rw . cél_könyvtár`.

Figyeljük meg, hogy a másolási módban nem használjuk az `f` kapcsolót, mivel nem jön létre biztonsági másolat. Ehelyett a régi könyvtárrendszer közvetlenül az új helyre másolódik. Ez a parancs sokkal könnyebben megjegyezhető, mint a `tar` megfelelője:

```
tar cf - . | (cd destination_directory; tar vpxf -)
```

Arra is figyeljünk, hogy *ezt soha nem akarjuk tenni*:

```
cd
mkdir test
pax -rw . test
```

A fenti példában beléptem a saját könyvtáramba, létrehoztam egy *test* nevű alkönyvtárat és elindítottam a másolást. Így a *test* alkönyvtárak végtelen sorát kaptam, mindegyikben a saját könyvtáram teljes tartalmával. Ha nem nyomom meg a CTRL-C billentyűkombinációt, a *pax* a végtelenségig folytatta volna a másolást; ebben az esetben a végtelen az a pont, amikor elfogy a lemezen a hely. Erre utal a `man pax` következő részlete is.

*„Figyelem, a célkönyvtár ne legyen valamelyik fájl műveletérték (operand) vagy valamelyik fájl műveletértéken belül kezdődő könyvtárszerkezet része, ugyanis ebben az esetben a másolás következménye beláthatatlan."*

A következő trükk azonban szépen és csaknem azonnal működik:

```
su
Password:
cd ~user1/big_project
mkdir ~user2/big_project
chown user2 ~user2/big_project
pax -rw . ~user2/big_project
```

Voilà, az egész *big_project* könyvtárszerkezet most már megtalálható a második felhasználó saját könyvtárában is. A másolást rendszergazdaként kell végrehajtanunk a saját könyvtárból, hogy elkerülhessük a végtelen ciklus csapdáját. Ha létre kell hozni a célkönyvtárat, mielőtt a másolást elkezdjük, használjuk a `chown` parancsot a könyvtár tulajdonosának megfelelő beállításához. A másolt könyvtárrendszer jogosultságainak kezelésével kapcsolatban a `man pax` nyújt további tájékoztatást.

Az `i` kapcsolóval a könyvtárrendszer felhasználói beavatkozást igénylő módú másolására is lehetőség nyílik:

```
pax -rw . ~/user2/big_project
```

A korábbi felhasználói beavatkozást igénylő módot használó példánkban látottakhoz hasonlóan a *pax* egyenként megjeleníti az összes fájlnevet, így kiválaszthatjuk, hogy mit másoljunk, és mit nevezzünk át közben.

## Növekményes mentések

Csináljunk valami hasznosat a `pax` paranccsal. Bemutatom hogyan hozhatunk létre növekményes mentési rendszert. Példánkban a genesis nevű felhasználó napi mentést végez a saját könyvtárának változásairól.

Először rendszergazdaként létrehozok egy könyvtárat a másolatoknak:

```
su
Password:
mkdir /usr/backups
```

Ezután létrehozok egy alkönyvtárat, és a genesis felhasználót teszem a tulajdonosává:

```
mkdir /usr/backups/genisis
chown genisis /usr/backups/genisis
```

Ezután kilépek a rendszergazdai héjból és genesis felhasználóként a saját könyvtáramba lépek át:

```
exit
cd
```

Majd teljes mentést készítek a saját könyvtáramról, amit a *hétfő* nevű fájlban helyezek el:

```
pax -wvf /usr/backups/genisis/hétfő .
```

Most, hogy egy teljes másolatom van, napi növekményes mentéseket végezhetek az aznapi változatok tárolására. Tehát, amikor befejezem a keddi munkát, a következő parancsot adom ki:

```
pax -wv -T 0000 -f /usr/backups/genisis/kedd .
```

Figyeljük meg a `-T` kapcsolót, ami az időt adja meg: ebben az esetben ez az éjfél (00:00). Ez arra utasítja a *pax*-ot, hogy csak az éjfél óta megváltozott fájlokat mentse, tehát az összes aznap módosított fájlt. Szerdán megismétlem ezt a parancsot, de a másolat nevét szerdára változtatom.

Ha van elég hely és szeretnénk egy hétnél hosszabb ideig tárolni a másolatokat, nevezzük a fájlokat másként, például: *aug1*, *aug2* stb. Így is érdemes hetente egy teljes mentést készíteni, amit a napi növekményes mentések követnek. Ha számít a tárhely nagysága, használjuk a `z` kapcsolót, így a másolatok tömörítettek lesznek. Felhívom a figyelmet arra is, hogy a `-T` kapcsoló sokkal választékosabban is használható, mint a példában, lásd `man pax`.

### Fájlok kihagyása helyreállításkor

A *file3* kivételével az összes fájlt a következő paranccsal állíthatjuk helyre:

```
pax -rvf ~/backup -c './file3'
```

A `c` kapcsoló a kivételt jelöli. A kivételeket jelölő mintát (ami ebben az esetben *a file3*) egyszeres zárójelek közé kell foglalni, vagy pontosan egyező mintát használjunk, mint a fenti példában (a *pax* számára ez a fájl a *./file3*, nem *file3*), vagy helyettesítő karaktert alkalmazzunk ilyen módon:

```
pax -rvf ~/backup -c '*file3'
```

Ha a minta elején használjuk a helyettesítő karaktert (*), mint a fenti példában, akkor minden olyan fájlt kihagyunk, aminek „file3" a vége – például *file3, myfile3, thatfile3.*

Az `n` kapcsolóval az is megadható, hogy melyik fájlokat állítsuk helyre. A következő példával csak a *file2* kerül helyreállításra:

```
pax -rvf ~/backup -n './file2'
```

Az `n` kapcsoló abban különbözik a `c` kapcsolótól, hogy csak az első fájlt állítja helyre, ami a mintának megfelel. Ez azt jelenti, hogy a következő parancs nem csomagolja ki a *file3, myfile3* és a *thatfile3* mindegyikét:

```
pax -rvf ~/backup -n '*file3'
```

Mivel a *file3* az első fájl, amelyre a minta illik, egyedül ez kerül helyreállításra.

A `c` és `n` kapcsolók mentéskor is használhatók: megadhatjuk, hogy melyik fájlokat akarjuk menteni, és mit akarunk kihagyni.

Vigyázat az `ext2` fájlrendszer legnagyobb mérete 2 GB, ebben az esetben használjuk a `split` programot!

## Kapcsolódó anyagok

- Az eredeti cikk a `http://www.onlamp.com/pub/a/bsd/2002/08/22/freeBSD_Basics.html` címen olvasható.
- A *pax* forráskód (az ast-open csomag része) a `http://www.research.att.com/sw/download/` címen érhető el.

# #40 Az indítórész mentése

## Mindig legyen kéznél egy másolat az indítórészről

Egy rendszerbetöltő program (például a LILO) telepítése több furcsasággal járhat, mint szeretnénk. Különösen, ha IDE-merevlemezeket használunk, könnyű bajba keveredni és olyan állapotba hozni a rendszert, hogy indítólemez nélkül nem lehet felélesztení.

Gyakori hiba az IDE-merevlemezen olyan lemezrészre telepíteni a rendszermagot, ami az első 1024 cilinderen túl nyúlik. A tünetek nagyon furcsák, mert a rendszer először szépen elindul, de amikor új rendszermagot telepítünk (miután egy ideig használtuk a gépet), a LILO hibát jelez, és a rendszer nem indul el a következő alkalommal. Ez nagyon zavarba ejtő jelenség, hiszen „eddig működött". Nagyon valószínű, hogy a rendszer telepítésekor a rendszermag az 1024 cilinderes határ alatt helyezkedett el. Telepítés után a rendszer összetevői és a felhasználói adatok kezdik feltölteni a helyet. Amikor a lemez körülbelül 500 MB (egyes BIOS-ok esetében 1 GB) adatot tartalmaz, az újonnan telepített rendszermag (legalább részben) az első 1024 cilinderen túl található – ami a BIOS számára nem elérhető a rendszer indításakor.

A LILO újabb változatai nem hajlandók ilyen rendszermagot telepíteni, de egyes régebbi változatok csak figyelmeztetést írnak ki, és folytatják a telepítést. Az egyik módja a feladat megoldásának az, ha a telepítési folyamat kezdetén létrehozunk egy kicsi (mondjuk 10 MB méretű) lemezrészt (mint */dev/hda1*, vagyis az első lemezrész) és */boot* néven fűzzük be. Ezután az új rendszermagokat mindig a */boot* alá telepítsük, így biztosan hozzáfér a BIOS, mert szükségszerűen az első 1024 cilinderen belül lesznek.

Akárhogyan nézzük is, a rendszerindító program telepítését nem lehet félválltól venni. Amikor beállítottuk a rendszerindító programot, és úgy működik, ahogyan szeretnénk, érdemes biztonsági mentést készíteni az egész indítórészről (bootblock).

Ez IDE-lemez esetében így fest:

```
dd if=/dev/hda of=bootsector.bin bs=512 count=1
```

SCSI-lemez esetében ilyen módon néz ki:

```
dd if=/dev/sda of=bootsector.bin bs=512 count=1
```

Esetleg menthetjük egyenesen hajlékonylemezre is:

```
dd if=/dev/hda of=/dev/fd0 bs=512 count=1
```

Nagyon figyeljünk oda az `if` és `of` megadására, mert felcserélésük az indítórész törléséhez vezethet. Valószínűleg jobb ötlet létrehozni a fájlt és utána másolni cserélhető adathordozóra, mint minden más fájlt. Fontos rendszeradatokat hajlékonylemezre bízni vicces elgondolás, de miután az első (vagy ezredik) lemez a legrosszabb pillanatban mond csütörtököt, az ember azt kívánja, bárcsak lett volna másik példánya.

Ezután a következő paranccsal bármikor könnyen visszaállíthatjuk az indítórészt, ha szükséges (miután az indítólemezről fölélesztettük a gépet):

```
dd if=bootsector.bin of=/dev/hda
```

Ha pedig az orvosi javallat ellenére is hajlékonylemezre mentettük:

```
dd if=/dev/fd0 of=/dev/hda
```

Ha SCSI-lemezt használunk, természetesen a *hda* helyére *sda*-t kell írnunk.

## Kapcsolódó anyag

- LILO Olvass el! fájlban (vagy más formátumú leírás a LILO telepítés *doc/* könyvtárában)

# #41 Fájlrendszerek egyes részeinek összehangolása az rsync paranccsal

## Tükrözzük az rsync paranccsal az SSH-n keresztül pontosan azt, amit akarunk, bármennyi kiszolgálóra

Az O'Reilly webes kiadványai számára olyan webkiszolgálótelepet építettünk ki, amely egy gép munkáját több kiszolgálóra osztja meg, növelve a sebességet és a megbízhatóságot. De hogyan hangolhatók össze egyes kiszolgálók fájlrendszerein az adtok?

Az egyik megoldás, ha NFS használatával közös állományrendszeren tartjuk az adatokat. Bár az NFS egyszerűen teszi lehetővé, hogy minden kiszolgáló számára egyidejűleg frissítsük az adatokat, néhány jelentős hátránya is akad. Az NFS teljesítménynövelő beállításait legtöbben a fekete mágia körébe sorolják. (Amikor működik, akkor nagyon jó, de amikor nem... Ezért állunk készenlétben, nem igaz?) De a monolitikus NFS kiszolgáló legnagyobb hátránya minden bizonnyal az, hogy egyetlen ponton az egész vállalkozás sebezhetővé válik. Ha az NFS kiszolgáló bármilyen okból nem érhető el vagy túlterhelt, annak minden tőle függő gép kárát látja.

Egy másik megközelítés szerint érdemes aszinkron módon frissítenünk a kiszolgálók tartalmát az *rsync* vagy hasonló eszköz segítségével. Most, amikor a merevlemezes tárolókapacitás ára minden eddiginél alacsonyabb, célszerű az adatokat gyorstárazásra a helyi gépre másolni, ilyen módon kiküszöbölhető az egyik sebezhető pont, és a fájlok a helyi lemezről sokkal gyorsabban szolgáltathatók, mint a hálózati tárolóról.

Nagyon egyszerűen beállítható az összes kiszolgálón az *rsync* futtatása `cron` feladatként:

```
rsync -ae ssh
master.machine.com:/usr/local/apache/htdocs/ \
   /usr/local/apache/htdocs/
```

Feltételezve, hogy az SSH-kulcsokat korábban már beállítottuk (lásd [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal), ez a parancs a helyi Apache dokumentumkönyvtárat frissíti a *master.machine.com* gépről

vett változattal titkosított SSH-kapcsolaton keresztül. Mindaddig, amíg a *master.machine.com* tartalmát frissítjük, a módosításokat hűen fogja tükrözni minden egyes kiszolgáló a parancs futása után.

E megközelítésnek az a legnagyobb hátránya, hogy a frissítéseknek aszinkron módon kell történnie. Ha ezt a parancsot a `cron` ötpercenként hajtja végre, akkor a helyi változat legfeljebb öt perccel lesz régebbi a központinál (átlagban körülbelül három perccel). Amennyiben az adott alkalmazás elviseli a nagy fájlrendszerekkel végzett `rsync` „egy perc múlva ott vagyok" természetét, ez a megoldás óriási nyereséget eredményez a sebesség és megbízhatóság terén. Fontos figyelmeztetés a `cron` feladatként futtatott `rsync` paranccsal kapcsolatban: vigyázzunk rá, hogy a feladat befejeződjön, mielőtt a következő indul. Ha a kiszolgáló le van terhelve, *zuhanórepülésbe kezdhet*, amikor az új `rsync` tovább növeli a terhelést, amitől a futó `rsync` hosszabb ideig tart, ami azt jelenti, hogy még fut, amikor a következő elindul, és ez megint növeli a terhelést.

Az *rsync* programnak néhány komoly előnye van a `tar`-hoz képest ebben az alkalmazáskörben. A `tar` az SSH-n keresztül (lásd [#37] Biztonsági mentés SSH-kapcsolaton keresztül a `tar` programmal) általában gyorsabb, amikor egy teljes könyvtárszerkezetet kell másolni, de az *rsync* sokkal gyorsabb, amikor csak néhány fájl változik a központi gépen. Először egy elemzést hajt végre, ezzel megállapítja, hogy mely fájlok változtak meg, és csak a változásokat viszi át.

De tegyük fel, hogy a webkiszolgálótelepnek még bonyolultabb feladatot kell megoldania. Mondjuk, meg akarjuk osztani a terhelést (lásd [#99] Terhelésmegosztás Apache RewriteMappel) két kiszolgálócsoport között: az alkalmazáskiszolgálók (amelyek az Apache `mod_perl` használatával a tartalomszolgáltatást végzik) és az első vonalat alkotó, pehelysúlyú Apache kiszolgálók, amik csak statikus adatokat szolgáltatnak (például képeket és fájlokat). Ebben az esetben pazarlás lenne az egész könyvtárfát minden kiszolgálóra átmásolni, hiszen az első vonalban lévő kiszolgálók csak néhány meghatározott fájlfajtát fognak elérhetővé tenni.

Az `rsync --exclude-from` kapcsolójával megadható egy fájl neve, amiből a program kiolvassa, hogy a fájlrendszer melyik részeit vegye figyelembe az összehangolás során. Hozzunk létre egy ehhez hasonló fájlt */usr/local/etc/balance.front* néven a statikus kiszolgálóknak:

```
### Stuff we definitely don't want to mirror.
#
- logs/

### the entire document root
#
+ /usr/
+ /usr/local/
+ /usr/local/apache/
+ /usr/local/apache/htdocs/
+ /usr/local/apache/htdocs/**

### user public_html directories
#
+ /home/
+ /home/*/
+ /home/*/public_html/
+ /home/*/public_html/**

# Images, archives, etc.
#
+ *.jpg
+ *.gif
+ *.png
+ *.pdf
+ *.mp3
+ *.zip
+ *.tgz
+ *.gz

# Exclude everything else.
#
- *
```

És hozzunk létre egy hasonló fájlt az alkalmazáskiszolgálóknak is:

```
### Stuff we definitely don't want to mirror.
#
- logs/
- *.tmp
- *.swp

### the entire document root
#
+ /usr/
+ /usr/local/
+ /usr/local/apache/
+ /usr/local/apache/htdocs/
+ /usr/local/apache/htdocs/**

# Exclude everything else.
#
- *
```

Most hozzunk létre két fájlt, amelyekben a különbözőfajta kiszolgálóinkat soroljuk fel, minden nevet külön sorban. A fájlok neve legyen */usr/local/etc/servers.front* és */usr/local/etc/serversback*.

Például a *servers.back* tartalma legyen ez:

```
tiberius
caligula
```

Ez pedig a *servers.back* fájlé:

```
augustus
claudius
germanicus
posthumous
castor
```

Végül ahelyett, hogy minden webkiszolgálón a `cron` démonnal közvetlenül hívnánk meg az *rsync* programot, próbáljuk ki ezt a kis Perl-kódot az eredeti változatot tartalmazó gépen:

**Lista: Balance-push.sh**

```
#!/bin/bash

#
# balance-push - Push content from the master
  ➥ server (localhost)
# to multiple front- and back-end servers,
  ➥ in parallel.
#

# $FRONT_END lists the servers that receive
  ➥ the front-end (e.g. static content) updates.
#
FRONT_END=$(cat /usr/local/etc/servers.front)

# $BACK_END lists the hosts that receive
  ➥ the full back-end (e.g. everything) updates.
#
BACK_END=$(cat /usr/local/etc/servers.back)

# $TARGET specifies the filesystem root
  ➥ on the remote host to push to.
# Normally, you want this to be /,
  ➥ unless you're doing testing.
#
TARGET=/

# $EXCLUDE specifies the prefix of the per-mode
  ➥ rsync exclude files.
# For example, if your exclude files are
  ➥ /usr/local/etc/balance.front and
# /usr/local/etc/balance.back, set this to
  ➥ "/usr/local/etc/balance".
# The per-mode extensions will be added.
#
EXCLUDE=/usr/local/etc/balance

# $LOCK_DIR specifies a path to put
  ➥ the lock files in.
#
LOCK_DIR=/var/tmp
```

```
### Ignore the shell functions behind the curtain. ###

PATH=/bin:/usr/bin:/usr/local/bin

lock (  ) {
local lockfile="$LOCK_DIR/balance.$1.lock"
if [ -f $lockfile ]; then
if kill -0 $(cat $lockfile); then
echo "$0 appears to be already running on $1."
echo "Please check $lockfile if you think this is
      ➥ in error."
exit 1
else
echo "$0 appears to have completed for $1 without
      ➥ cleaning up its lockfile."
fi
fi
echo $$ > $lockfile
}

unlock (  ) {
rm -f $LOCK_DIR/balance.$1.lock
}

push_files (  ) {
local mode=$1 host=$2

if [ ! "$mode" -o ! -r "$EXCLUDE.$mode" ]; then
echo "$0 $$: mode unset for $host!"
return
fi

if [ ! "$host" ]; then
echo "$0 $$: host unset for push $mode!"
return
fi

lock $host

rsync --archive --rsh=ssh --delete \
--ignore-errors --whole-file \
--exclude-from="$EXCLUDE.$mode" / ${ host}
➥ :${ TARGET}
```

```
unlock $host
}

push_tier ( ) {
local mode=$1 host_list=$2

for host in $host_list; do
$SHELL -c "push_files $mode $host" &
done
}

export -f lock unlock push_files
export TARGET EXCLUDE LOCK_DIR PATH

[ "$FRONT_END" ] && push_tier front "$FRONT_END"
[ "$BACK_END" ] && push_tier back "$BACK_END"

#
# Fin.
#
```

Ez a parancsfájl (legyen a neve *balance-push*) vezérli az `rsync` futását, biztosítva, hogy ne legyen „átfedés" a futások között. Elküldi a megfelelő fájlokat mindkét csoportnak, attól függően, amit a */usr/local/etc/* alatt adunk meg. Ha olyan kiszolgálót talál, amelyik még nem fejezte be az előző `rsync` végrehajtását, akkor kihagyja ezt a kiszolgálót, amíg nem végzett (és erről levélben értesít a `cron` `MAILTO` szolgáltatásával).

A központi gép terhelése valószínűleg megnő, attól függően, hogy hány kiszolgálót kell összehangolni (mivel minden kiszolgálónak egy `rsync` és egy `ssh` folyamatra is szüksége van). De gyakorlati szempontból, egy naponta találatok millióit kiszolgáló hálózaton, az egyes webkiszolgálókra jutó terhelés elhanyagolható.

# #42 Önműködő pillanatfelvétel-típusú növekményes mentések az rsync programmal

## Használjuk az rsync programot gyors, kis méretű és biztos pillanatfelvételek készítésére

Következzen egy egyszerű módszer a pillanatfelvétel-szerű mentések készítésére Linux-kiszolgálókon. A pillanatfelvétel felső kategóriás, nagy megbízhatóságú fájlkiszolgálókon szokásos mentési eljárás, napi több, teljes értékű (tulajdonos/engedély szintig) mentés *illúzióját* kelti a teljes mentés hely- és időigénye nélkül. Minden pillanatfelvétel csak olvasható és közvetlenül hozzáférhető a felhasználók számára különleges rendszerkönyvtárak formájában.

Miután nagy állományrendszerekről teljes mentést készíteni időrabló és drága folyamat, általános módszer, hogy teljes mentést csak hetente vagy havonta készítünk, a többi napon pedig a változásokat tároljuk. Ezt a módszert nevezzük „növekményes" mentésnek, és többek között a jó öreg *tar* és *dump* segédprogramok is támogatják.

A GNU fájlkezelő programok közül a `cp` parancs a `-l` kapcsolóval kiadva (közvetlen) hivatkozásokat hoz létre másolat helyett (bár nem készít hivatkozásokat könyvtárakra, ez így helyes, de eltűnődhetünk rajta, hogy miért). A `cp` parancs következő hasznos kapcsolója a `-a` (mint archive); ennek hatására végighalad az alkönyvtárakon, és megőrzi a tulajdonost, a hozzáférési jogosultságokat, valamint az időadatokat.

Együtt a `cp-al` látszólag a könyvtárrendszer teljes másolatát készíti el, de valójában ez csak egy illúzió, ami szinte semmi helyet nem foglal. Ha a másolaton végezhető műveleteket fájlok hozzáadására és eltávolítására korlátozzuk, és soha nem változtatjuk meg a fájlokat, akkor a hiánytalan másolat illúziója teljes lesz. A felhasználó szemében az egyetlen különbség az, hogy az illúziómásolat szinte semmi helyet nem foglal, és szinte semennyi időbe sem telik létrehozni.

Az `rsync` és a `cp` összekapcsolásával látszólag teljes mentéseket végezhetünk egy fájlrendszerről anélkül, hogy több lemeznyi helyet használnánk el.

Például:

```
rm -rf backup.3
mv backup.2 backup.3
mv backup.1 backup.2
cp -al backup.0 backup.1
rsync -a --delete source_directory/ backup.0/
```

Ha a fenti parancsokat mindennap futtatjuk, akkor a *backup.0*, *backup.1*, *backup.2* és *backup.3* külön-külön mind a *source_directory/* teljes másolatának tűnik, amilyen az ma, tegnap, tegnapelőtt, két napja, illetve három napja volt, teljes egészében – azzal az egy különbséggel, hogy a hozzáférési jogosultságok és tulajdonosok a régi pillanatfelvételekben az új értékre állnak be (köszönet éret W. J. Schultznak, aki erre felhívta a figyelmet). Valójában a helyigény megegyezik a *source_directory/* pillanatnyi méretével plusz az elmúlt három nap összes változtatásának méretével – pontosan úgy, mintha a *dump* vagy a `tar` készítene egy teljes, majd növekményes mentéseket.

Ez a módszer *sokkal* jobb a hálózati mentésekhez, mivel elegendő egyszer elkészíteni a teljes másolatot, a heti teljes mentések helyett. Ezután már csak a változásokat kell átmásolni. Szalagra sajnos nem lehet menteni a `rsync` paranccsal – ehhez továbbra is a `dump` vagy a `tar` szükséges.

Ha kéznél van egy gép, akár nagyon kis teljesítményű is lehet, használhatjuk a biztonsági mentésekre szakosodott kiszolgálóként. Tartsuk elszigetelve, lehetőleg más helyen, mint az eredeti adatokat tartalmazó gépet, másik szobában vagy akár másik épületben. Tiltsunk le rajta minden elképzelhető távoli szolgáltatást, és csak a forrásgép erre a célra szánt hálózati illesztőjéhez csatlakoztassuk.

Ezután erről a gépről SSH-n keresztül végezhetjük a mentéseket az *rsync* programmal (lásd [#41] Fájlrendszerek egyes részeinek összehangolása az `rsync` paranccsal), és a másolatokat csak olvasható NFS-en keresztül biztosíthatjuk az eredeti gépnek. Így a felhasználók maguk is elérhetik a pillanatfelvételeket (nincs szükség a rendszergazda beavatkozására), és véletlenül sem törölhetik le vagy változtathatják meg őket.

Ezt már elfogadhatjuk „elég jó" védelemnek, de aki (bölcsen) üldözési mániában szenved, építsen két másolatkiszolgálót. Azután meggyőződhet róla, hogy legalább az egyik mindig le van állítva.

### Kiterjesztések: pillanatfelvételek óránként, naponta, hetente

Egy kis ügyeskedéssel a többszintű pillanatfelvételek rotációját is beállíthatjuk. Például az én gépemen a négy legutóbbi „óránkénti" (valójában négyóránként rögzített) és a három legutóbbi „napi" (ami mindennap éjfélkor kerül mentésre) pillanatfelvételt tárolom. A szükségleteknek és a rendelkezésre álló tárolóhelynek megfelelően lehet heti vagy akár havi pillanatfelvételeket is tárolni.

Létezik egy parancsállományom, ami négyóránként elkészíti és forgatja a pillanatfelvételeket, és egy másik, amelyik naponta egyszer fut. Szükségtelen az `rysnc`-et használni a felsőbb szintű pillanatfelvételekhez; a `cp -al` előállítja a megfelelő óránkénti másolatból.

A pillanatfelvételek önműködő futtatásához a következő bejegyzéseket tettem a fő *crontab* állományba:

```
0 */4 * * * /usr/local/bin/make_snapshot.sh
0 13 * * * /usr/local/bin/daily_snapshot_rotate.sh
```

A *make_snapshot.sh* parancsfájlt négy óránként, illetve a *daily_snapshot_rotate.sh* parancsfájlt pedig mindig 13:00 órakor futtatják. Alább következnek a parancsfájlok:

### Lista: make_snapshot.sh

```
#!/bin/bash
# ----------------------------------------------
# mikes handy rotating-filesystem-snapshot utility
# ----------------------------------------------
# RCS info: $Id: ch03,v 1.21 2003/03/07 20:16:38
ldolby Exp davidf $
# ----------------------------------------------
# this needs to be a lot more general,
  ➥ but the basic idea is it makes
# rotating backup-snapshots of
  ➥ /home whenever called
```

```
# ------------------------------------------------

# ----- system commands used by this script ------
ID=/usr/bin/id;
ECHO=/bin/echo;

MOUNT=/bin/mount;
RM=/bin/rm;
MV=/bin/mv;
CP=/bin/cp;
TOUCH=/bin/touch;

RSYNC=/usr/bin/rsync;


# ---------------- file locations -----------------

MOUNT_DEVICE=/dev/hdb1;
SNAPSHOT_Rw=/root/snapshot;
EXCLUDES=/usr/local/etc/backup_exclude;


# --------------- the script itself --------------

# make sure we're running as root
if (( `$ID -u` != 0 )); then {
➥ $ECHO "Sorry, must be root. Exiting..."; exit; }
➥ fi

# attempt to remount the Rw mount point as Rw;
  ➥ else abort
$MOUNT -o remount,rw $MOUNT_DEVICE $SNAPSHOT_Rw ;
if (( $? )); then
{
$ECHO "snapshot: could not remount $SNAPSHOT_Rw
       ➥ readwrite";
exit;
}
fi;

# rotating snapshots of /home (fixme: this should
  ➥ be more general)
```

```
# step 1: delete the oldest snapshot,
  ➥ if it exists:
if [ -d $SNAPSHOT_Rw/home/hourly.3 ] ; then \
$RM -rf $SNAPSHOT_Rw/home/hourly.3 ; \
fi ;

# step 2: shift the middle snapshots(s)
  ➥ back by one, if they exist
if [ -d $SNAPSHOT_Rw/home/hourly.2 ] ; then \
$MV $SNAPSHOT_Rw/home/hourly.2
➥ $SNAPSHOT_Rw/home/hourly.3 ; \
fi;
if [ -d $SNAPSHOT_Rw/home/hourly.1 ] ; then \
$MV $SNAPSHOT_Rw/home/hourly.1
➥ $SNAPSHOT_Rw/home/hourly.2 ; \
fi;

# step 3: make a hard-link-only (except for dirs)
  ➥ copy of the latest snapshot,
# if that exists
if [ -d $SNAPSHOT_Rw/home/hourly.0 ] ; then \
$CP -al $SNAPSHOT_Rw/home/hourly.0
➥ $SNAPSHOT_Rw/home/hourly.1 ; \
fi;

# step 4: rsync from the system into
  ➥ the latest snapshot (notice that
# rsync behaves like cp --remove-destination
  ➥ by default, so the destination
# is unlinked first. If it were not so, this would
  ➥ copy over the other
# snapshot(s) too!
$RSYNC \
-va --delete --delete-excluded \
--exclude-from="$EXCLUDES" \
/home/ $SNAPSHOT_Rw/home/hourly.0 ;

# step 5: update the mtime of hourly.0 to reflect
  ➥ the snapshot time
$TOUCH $SNAPSHOT_Rw/home/hourly.0 ;

# and thats it for home.
```

```
# now remount the Rw snapshot
  ➥ mountpoint as readonly

$MOUNT -o remount,ro $MOUNT_DEVICE $SNAPSHOT_Rw ;
if (( $? )); then
{
$ECHO "snapshot: could not remount $SNAPSHOT_Rw
       ➥ readonly";
exit;
}  fi;
```

Az rsync számára létrehoztam egy kivétellistát, melynek csupán az a célja, hogy megakadályozza különféle szemét mentését, amilyen például a webböngésző gyorstár, ami gyakran változik (tehát sok helyet foglalna el a pillanatfelvételekben), de az sem volna veszteség, ha véletlenül megsemmisülne.

### Lista: daily_snapshot_rotate.sh

```
#!/bin/bash
# ------------------------------------------------
# mikes handy rotating-filesystem-snapshot utility:
daily snapshots
# ------------------------------------------------
# RCS info: $Id: daily_snapshot_rotate.sh,v 1.2
2002/03/25 21:53:27 mrubel Exp $
# ------------------------------------------------
# intended to be run daily as a cron job when
hourly.3 contains the
# midnight (or whenever you want) snapshot; say,
13:00 for 4-hour snapshots.
# ------------------------------------------------

# ------ system commands used by this script ------
ID=/usr/bin/id;
ECHO=/bin/echo;

MOUNT=/bin/mount;
RM=/bin/rm;
MV=/bin/mv;
cp=/bin/cp;
```

```
# --------------- file locations -----------------

MOUNT_DEVICE=/dev/hdb1;
SNAPSHOT_Rw=/root/snapshot;

# -------------- the script itself ---------------

# make sure we're running as root
if (( `$ID -u` != 0 )); then {
➥ $ECHO "Sorry, must be root. Exiting..."; exit; }
➥ fi

# attempt to remount the Rw mount point as Rw;
  ➥ else abort
$MOUNT -o remount,rw $MOUNT_DEVICE $SNAPSHOT_Rw ;
if (( $? )); then
{
$ECHO "snapshot: could not remount $SNAPSHOT_Rw
       ➥ readwrite";
exit;
}
fi;


# step 1: delete the oldest snapshot, if it exists:
if [ -d $SNAPSHOT_Rw/home/daily.2 ] ; then \
$RM -rf $SNAPSHOT_Rw/home/daily.2 ; \
fi ;

# step 2: shift the middle snapshots(s) back
  ➥ by one, if they exist
if [ -d $SNAPSHOT_Rw/home/daily.1 ] ; then \
$MV $SNAPSHOT_Rw/home/daily.1
➥ $SNAPSHOT_Rw/home/daily.2 ; \
fi;
if [ -d $SNAPSHOT_Rw/home/daily.0 ] ; then \
$MV $SNAPSHOT_Rw/home/daily.0
➥ $SNAPSHOT_Rw/home/daily.1; \
fi;

# step 3: make a hard-link-only (except for dirs)
  ➥ copy of
```

```
# hourly.3, assuming that exists, into daily.0
if [ -d $SNAPSHOT_Rw/home/hourly.3 ] ; then \
$cp -al $SNAPSHOT_Rw/home/hourly.3
➥ $SNAPSHOT_Rw/home/daily.0 ; \
fi;

# note: do *not* update the mtime of daily.0;
  ➥ it will reflect
# when hourly.3 was made, which should be correct.

# now remount the Rw snapshot mountpoint as readonly

$MOUNT -o remount,ro $MOUNT_DEVICE $SNAPSHOT_Rw ;
if (( $? )); then
{
$ECHO "snapshot: could not remount $SNAPSHOT_Rw
       ➥ readonly";
exit;
}  fi;
```

### Példa az ls -l /snapshot/home parancs eredményére

```
drwxr-xr-x 12 root root 4096 Mar 28 00:00 daily.0
drwxr-xr-x 12 root root 4096 Mar 27 00:00 daily.1
drwxr-xr-x 12 root root 4096 Mar 26 00:00 daily.2
drwxr-xr-x 12 root root 4096 Mar 28 16:00 hourly.0
drwxr-xr-x 12 root root 4096 Mar 28 12:00 hourly.1
drwxr-xr-x 12 root root 4096 Mar 28 08:00 hourly.2
drwxr-xr-x 12 root root 4096 Mar 28 04:00 hourly.3
```

Figyeljük meg, hogy a */snapshot/home/* minden alkönyvtára a */home* teljes lenyomata abban az időpontban, amikor a pillanatfelvétel készült. A könyvtárak hozzáférési jogosultságaiban látható w ellenére senki – még a rendszergazda sem – írhat ebbe a könyvtárba; csak olvasható módban van befűzve.

# #43 A lenyomatfájlok és a CDR/CD-RW-lemezek használata

## A parancssorból gyerekjáték a CD-lenyomatokat kezelni

Nagy számú grafikus CD-író segédprogram létezik Linuxhoz. Ezek legtöbbje egyszerűen egy kezelőfelület, ami meghívja a lenyomatkészítés (ISO) és a CD-írás munkáját valójában elvégző programokat. A grafikus eszközök nem sokat segítenek, ha olyan kiszolgálókon dolgozunk, amikhez nem kapcsolódik konzol, hacsak nem futtatjuk az X-et távolról, mondjuk az SSH-n keresztül (lásd [#70] X az SSH alatt). Ha az *mkisofs* és a *cdrecord* programok megtalálhatók a gépen, akkor a CD-lenyomatokkal végzett munka a parancssorban meglehetősen egyszerű.

Ha valaki még soha sem hallotta volna a lemezlenyomat (ISO images) kifejezést, ez egy ISO9660 fájlrendszert tartalmazó, CD-re írható állomány. Az ISO9660 szabvány (néhány kiterjesztéssel) a CD-ROM-ok általánosan használt adattárolási formátuma.

Lenyomatfájlt az `mkisofs` paranccsal készíthetünk:

```
mkisofs -r /home/rob/ > /tmp/rob-home.iso
```

A `-r` kapcsoló azt jelenti, hogy a létrejövő lenyomatfájl Rock Ridge kiterjesztést is tartalmazzon. Ez azt jelenti, hogy a hosszú fájlnevek és a fájlok hozzáférési jogosultságai helyesen jelennek meg, amikor a lemezt Rock Ridge kiterjesztést támogató rendszeren fűzik be. A Linux nagyszerűen támogatja az Rock Ridge-t, és a `-r` kapcsoló nagyon megkönnyíti az életet, amikor linuxos használatra szánt lemezt készítünk. Bárki készíthet lenyomatot olyan fájlokból, amikre olvasási jogosultsággal bír; az `mkisofs` futtatásához nem szükséges rendszergazdának lenni. Az itt szereplő többi parancshoz viszont rendszergazdai jogosultságok kellenek.

Jegyezzük meg, hogy az `mkisofs` a parancssorban megadott könyvtárak tartalmát tárolja és nem magukat a könyvtárakat. A fenti példában a lenyomat / könyvtárához a */home/rob/* könyvtárat rendeljük hozzá, így ennek a könyvtárnak a tartalma tölti meg a CD főkönyvtárát.

Egy adattároló CD-ről így készíthetünk lenyomatot:

```
# dd if=/dev/cdrom of=image.iso
```

Ez létrehoz egy lenyomatot a teljes CD-ről, mind a maga fenséges 650+ MB méretével, úgyhogy győződjünk meg róla, hogy van-e elég hely, mielőtt kipróbáljuk. Ezt a folyamatot esetleg gyorsíthatjuk a `dd` parancs `bs` kapcsolójával:

```
# dd if=/dev/cdrom of=image.iso bs=10k
```

A legjobb beállítás általában a meghajtó és az IDE illesztőfüggvénye, úgyhogy próbáljunk ki néhány értéket a legjobb beállítás kikísérletezéséhez.

A lenyomatot (amit `mkisofs` vagy `dd` segítségével létrehoztunk) így fűzhetjük be:

```
# mkdir /mnt/iso
# mount -o loop,ro -t iso9660 ./image.iso /mnt/iso
```

Ha a `mount` parancs a „Could not find any loop device" választ adja, lehet, hogy be kell töltenünk a hurokeszközvezérlő-modult (loopback driver):

```
# modprobe loop
```

Amikor a lenyomatot befűzzük, beléphetünk a */mnt/iso* könyvtárba és körülnézhetünk a lenyomatban lévő fájlrendszeren belül. Ha valami gond van vele, egyszerűen válasszuk le az `umount /mnt/iso` paranccsal, töröljük a lenyomatot, és próbálkozzunk újra.

A lenyomatot a következő alakú paranccsal írhatjuk lemezre:

```
# cdrecord -v speed=12 dev=0,0,0 -data image.iso
```

A *speed*= beállításban a saját CD-írónk sebességét adjuk meg (vagy kevesebbet). Ha egy CD-RW-lemezt írás előtt akarunk törölni, használjuk a *blank*= lehetőséget:

```
# cdrecord -v speed=12 dev=0,0,0 blank=fast
  ➥ -data image.iso
```

A CD-írót működésre bírni Linux alatt már nem olyan nehéz, mint régebben volt, hála az `ide-scsi`-meghajtónak. Ez egy rendszermagmodul, ami az IDE (és egyéb) CD-írókat úgy tünteti föl, mintha SCSI-írók lennének, amit sokkal könnyebb programozni.

### Kapcsolódó anyag

- A CD-író beüzemeléséhez segítséget nyújt a CD Writing HOWTO, ami a Linux Documentation Project `http://www.tldp.org/HOWTO/CD-Writing-HOWTO.html` webcímén érhető el.

## #44 CD-írás lenyomatfájl készítése nélkül

### Készítsünk CD-t egy másik CD-ről, a fájlrendszerből vagy akár egy HTTP-letöltésből

Egy CD megírásának legbiztosabb módja, ha először létrehozzuk a lenyomatot, azután kiírjuk (lásd [#43] A lenyomatfájlok és a CDR/CD-RW-lemezek használata). De néha nincs hely (vagy idő) a közbeeső lépésre. Ez legjobban akkor oldható meg, ha a forrás-CD és az író külön eszközláncon van (például első és második IDE-illesztő, vagy IDE és SCSI).

Valós időben ilyen módon készíthetünk másolatot egy CD-ről:

```
# dd if=/dev/hdb | cdrecord -v speed=12 dev=0,0,0
  ➥ fs=8m -data -
```

A - (mínuszjel) azt jelenti, hogy a *cdrecord* az adatsávot a szabályos bemenetről veszi, nem fájlból. A `dd` parancs kimenete táplálja a csővezeték végén a *cdrecord* programot az első `ide` illesztőn lévő másodlagos (slave) meghajtóban (*/dev/hdd*) lévő CD adataival.

Az `fs=8m` az író FIFO-t állítja kicsit nagyobbra, hogy kiküszöböljük az adatfolyam pillanatnyi megakadásának hatását. Amíg a busz megállja a helyét (és a gép nincs különösebben leterhelve semmi mással) ez a módszer nagyon jól működik.

Hasonlóképpen arra sincs igazán szükség, hogy lenyomatot készítsünk, mielőtt a fájlrendszerből adatokat írnánk ki. Próbáljuk így:

```
# mkisofs -r /home/ftp/ | cdrecord -v speed=12
  ➥ dev=0,0,0 fs=8m -data -
```

Éppúgy, mint a `dd`, az `mkisfs` alapértelmezés szerint szintén a szabályos kimenetre ír. Ez kerül azután a *cdrecord* bemenetére, így a lenyomat elkészítésével egy időben ki is íródik a lemezre. Ezzel elkerülhető, hogy a lenyomatfájlból egy példányt a merevlemezen kelljen tartanunk. Azonban figyeljünk arra, hogy az írható lemez méreténél (valahol 650 és 700 MB között) nagyobb adatforrás esetén kávéscsésze alátét lesz az eredmény, nem pedig használható lemez.

Először nézzük meg a `du` paranccsal, hogy mennyi helyre lesz szükség:

```
# du -hs /home/ftp/
412M /home/ftp
```

Tökéletes. Egy 412 MB-os lenyomat könnyűszerrel ráfér a lemezre.

Bármi szerepelhet a csővezeték bal oldalán, ami egy lenyomatfájlt szabályos kimenetre tud küldeni. Mit szólnánk egy közvetlen CD-íráshoz a hálózaton keresztül?

```
root@catlin:~# mkisofs -r backup/ \
  | ssh florian "cdrecord -v speed=12 dev=0,0,0
    ➥ fs=8m -data -"
```

Vagy küldjük a helyi meghajtóról a lenyomatfájlt közvetlenül egy távoli gépen lévő íróra a hálózaton keresztül?

```
root@catlin:~# dd if=/dev/cdrom \
   | ssh florian "cdrecord -v speed=12 dev=0,0,0
     ➥ fs=8m -data -"
```

Esetleg akár egyetlen lépésben letölthetünk és kiírhatunk egy lenyomatot:

```
# curl http://my.server.com/
   ➥ slackware-8.1-install.iso | cdrecord -v
   ➥ speed=0 dev=0,0,0 fs=8m -data -
```

Nem javasolnám, hogy hálózati CD-írással vezeték nélküli kapcsolaton próbálkozzunk; ehhez a munkához nem árt egy szép, megbízható 100 Mb/s-os kapcsolat. A letöltős példa viccesen nézhet ki, de jól mutatja a Unix-csővezeték hatékonyságát: bármelyik program összeköthető szinte bármelyik másikkal, és így olyasvalami jön létre, amire valószínűleg soha nem gondolt egyik program megalkotója sem.

# Hálózatkezelés

## 45–53. trükkök

A régi szép időkben a rendszergazda feladata az volt, hogy kitalálja, miként lehetne rábírni a hálózatra kötött számítógépeket, hogy szót értsenek egymással. Mostanában viszont munkaidejének nagy részét arra fordítja, milyen módon korlátozza a hálózati hozzáférést, távol tartandó a nem kívánatos betolakodókat, ugyanakkor átengedje a szabályos adatforgalmat.

Szerencsére a Netfilter nagyon rugalmas felületet ad a rendszermagnak a hálózatkezeléssel kapcsolatos döntésekhez. Az `iptables` parancs használatával létre lehet hozni összetett és nagyon rugalmas hozzáférést szabályozó házirendet (access policy). Ez a csomagokat nemcsak kapu (port), csatolóeszköz (interface) és MAC-címek szerint képes összeilleszteni, de a csomagok adattartalma és sűrűségük alapján is. Ezek az adatok segíthetnek megtalálni a különféle támadási kísérleteket, a kapuelárasztástól (port flood) a vírusokig.

A felhasználók kizárása korántsem okoz akkora örömet, mint az összekapcsolásuk. A számítógép-hálózat végső soron azt a célt szolgálja, hogy lehetővé tegye az emberek számára a kapcsolattartást. Meg fogunk vizsgálni néhány szokatlan módszert a hálózati forgalom

szabályozására, a távoli kaputovábbítástól kezdve az IP-alagutazás különböző formáival bezárólag. Mire megismerkedünk az IP-tokozással – és az olyan felhasználói térben működő alagutakkal, mint amilyen például a `vtun` –, addigra meg fogjuk érteni, hogyan lehet hálózatot kialakítani az internetre építve, ami számtalan váratlan, ám meglepően hasznos szolgáltatásra képes.

## #45 Tűzfal létrehozása tetszőleges kiszolgáló parancssorából

### Az iptables adta lehetőségek kihasználásához nincs szükség kijelölt kiszolgálóra

A Netfilter alapú tűzfal – amely a 2.4-es és azt követő rendszermagokban érhető el – nagyon rugalmas, parancssori eszközökkel irányított tűzfalkezelést tesz lehetővé. Szükség lehet némi időre, amíg valaki megszokja az `iptables` használatát, cserébe nagyon kifejező írásmódjával (syntax) teszi lehetővé összetett és remélhetőleg hasznos tűzfalszabályok megalkotását.

A csomagszűrő szolgáltatás abban az esetben is hasznos lehet, ha a gép valójában nem tűzfal, vagyis csupán egy hálózati kártya van beleszerelve, és nem védelmez további gépeket. Tegyük fel, hogy ehhez a géphez Telnet-hozzáférést szeretnénk engedélyezni – pusztán arra az esetre, ha az SSH-val vagy annak könyvtáraival mégis történne valami –, de nem áll szándékunkban az engedélyt úgy kiadni, hogy az egész világháló bármely pontjáról be lehessen lépni. Erre a célra használhatnánk egy TCP-burkolót (wrapper), benépesítve a */etc/hosts.allow* és */etc/hosts.deny* állományokat, és megfelelő módon beállítva a */etc/inetd.conf* állományt.

Az `iptables`-t az alábbihoz hasonló parancssorral is használhatjuk:

```
iptables -A INPUT -t filter -s ! 208.201.239.36
➥ -p tcp --dport 23 -j DROP
```

A legtöbb ember általában a „tiszta" oldalon korlátlan hozzáférést szeretne engedélyezni, a kényes gépeknél pedig megakadályozni a hozzáférést és mindenki másnak valamiféle köztes megoldást kiala-

kítani. Az alábbiakban bemutatunk egy egyidejűleg használható fehér lista (whitelist), fekete lista (blacklist) és engedélyezett kapukezelési rendet.

```
#!/bin/sh
#
# A simple firewall initialization script
#
WHITELIST=/usr/local/etc/whitelist.txt
BLACKLIST=/usr/local/etc/blacklist.txt
ALLOWED="22 25 80 443"

#
# Drop all existing filter rules
#
iptables -F

#
# First, run through $WHITELIST, accepting all
  ➥ traffic from the hosts and networks
# contained therein.
#
for x in `grep -v ^# $WHITELIST |
➥ awk '{ print $1} '`; do
echo "Permitting $x..."
iptables -A INPUT -t filter -s $x -j ACCEPT
done

#
# Now run through $BLACKLIST, dropping all traffic
  ➥ from the hosts and networks
# contained therein.
#
for x in `grep -v ^# $BLACKLIST |
➥ awk '{ print $1} '`; do
echo "Blocking $x..."
iptables -A INPUT -t filter -s $x -j DROP
done

#
# Next, the permitted ports: What will we accept
  ➥ from hosts not appearing
```

```
# on the blacklist?
#
for port in $ALLOWED; do
echo "Accepting port $port..."
iptables -A INPUT -t filter -p tcp
➥ --dport $port -j ACCEPT
done

#
# Finally, unless it's mentioned above, and it's an
  ➥ inbound startup request,
# just drop it.
#
iptables -A INPUT -t filter -p tcp --syn -j DROP
```

Győződjünk meg róla, hogy a `$ALLOWED` változóban az összes beengedni kívánt kaput felsoroltuk-e. Ha a felsorolásból kimaradna a 22-es kapu, nem lehetne az SSH-val bejutni a gépre.

A */usr/local/etc/blacklist* állományt az alábbihoz hasonló gépneveket és hálózatokat jelölő megjegyzések fogják megtölteni:

```
1.2.3.4 # Portscanned on 8/15/02
7.8.9.0/24 # Who knows what evil lurks therein
r00tb0y.script-kiddie.coop
  ➥ # $0 s0rR33 u 31337 h4x0r!
```

Hasonlóképpen a */usr/local/etc/whitelist.txt* állományban soroljuk fel a „jó embereket", akik számára – bármit is mondanak az egyéb szabályok – a hozzáférés megengedett:

```
11.22.33.44 # munkaállomásom
208.201.239.0/26 # helyi hálózat
```

Minthogy kizárólag a kettős kereszttől (#) eltérő karakterrel kezdődő sorokat vesszük figyelembe, így szükség szerint akár teljes sort vagy sorokat is megjegyzéssé alakíthatunk. A héjprogram legközelebbi futtatása során az ilyen módon megjegyzéssé alakított sorokat már figyelmen kívül hagyja. A munka kezdetén a már létező szűrési parancsok törléséhez adjuk ki az `iptables -F` utasítást, hogy egyszerűen

tudjuk végrehajtani parancsot, ha a *blacklist.txt*, vagy *whitelist.txt*, vagy a kapukijelöléseket tartalmazó állomány tartalmát módosítjuk.

Vegyük figyelembe, hogy ez a héjprogram egyelőre csak a TCP/IP-kapcsolatokat teszi lehetővé. Abban az esetben, ha UDP, ICMP vagy más protokollok használatát is támogatnunk kell, akkor a hurokeszközhöz (loop) tartozó `$ALLOWED` állományához hasonlóan futtassuk le még egyszer a programot, de ezúttal illesszük be a kiegészítő protokollokat és kapukat is. Például:

```
passing -p udp or -p icmp to iptables
```

Legyünk óvatosak a whitelist használatával, ugyanis ebben a listában szereplő bármely IP-cím vagy hálózat jogosultságot kap a gépen levő összes kapuhoz való hozzáférésre. Bizonyos körülmények között egy okos gonosztevő képes lehet rá, hogy hamisított, látszólag a listában szereplő címek egyikéről származó csomagokat küldjön, ha idejekorán előre vagy logikai következtetés útján ki tudja találni milyen címek szerepelnek a fehér listán (whitelist). Az ilyen támadást nehéz véghezvinni, de nem lehetetlen. Ha (bölcsen) üldözési mániánk van, valószínűleg csak olyan fehér listás címeket szeretnénk engedélyezni, amelyek nem használnak internetes útválasztót, hanem csak a belső hálózaton közlekednek.

Az új tűzfalszabályok kidolgozása közben rendkívül fontos, hogy mindig legyen konzol-hozzáférésünk, az IP Tables programmal nem lehet kizárni magunkat a konzolból. Ha az IP Tables programmal végzett munka közben már nagyon összekuszálódtak a szálak az `iptables -L` paranccsal mindig készíthetünk listát a beállított szabályokról, valamint a `-iptables -F` parancs kiadása után mindent tiszta lappal újra kezdhetünk. (Fontos! Ha rendszerünk alapértelmezett viselkedése a tiltás – deny –, ezzel a parancssal minden hálózati forgalmat azonnal kitiltunk, esetleg saját magunkat is! – Lektor)

Ha úgy tűnik, hogy az `iptables -L` nem működik, próbáljuk meg az alábbi parancs alkalmazását:

```
iptables -L -n
```

Ennek segítségével megtekinthetjük a tűzfalszabályokat névfeloldás nélkül, ugyanis a tűzfalszabályok megakadályozhatják a DNS-kéréseket.

Az egyszerű `iptables` szűréssel sok mindent el lehet érni, ugyanakkor az `iptables` sokkal több lehetőséget tartogat számunkra a célcsomagok szűrésénél.

### Kapcsolódó anyagok

- Netfilter HOGYAN a `www.netfilter.org` címen
- [#47] IP Tables tippek és trükkök

## #46 Egyszerű IP-álcázás

### NAT telepítése átjáróra tíz másodperc alatt

Ha egyetlen belső hálózattal rendelkezünk, amelyhez szükség van egy IP-címes megosztott internetkapcsolatra, akkor az átjárón IP-álcázást kell alkalmaznunk. Szerencsére az IP Tables programmal ez csak egy egyszerű kétsoros utasítás:

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
# iptables -t nat -A POSTROUTING -o $EXT_IFACE
  ➥ -j MASQUERADE
```

Az utasításban a `$EXT_IFACE` változó az átjáró külső hálózati kártyáját jelöli. Ettől kezdve az átjáró másik kártyáján levő hálózatra kapcsolt gépek „ki tudnak jutni" az internetre. A világháló felől nézve a teljes forgalom az átjáró külső címe felől érkezik.

Volt idő, amikor a külső hálózaton levő gonosztevők miatt kellett aggóni, akik hamisított csomagokat küldtek az átjárónak, úgy tüntetve fel magukat, mintha a belső hálózatról származnának. Ezeket a csomagokat a rendszermag kötelező jelleggel álcázta, és úgy hagyták el a hálózatot, mintha maguk is a „tisztességes" adatforgalom részeit képeznék. Ez lehetővé tette bárki számára, hogy látszólag belső hálózatról indított támadásokat kezdeményezzen, amivel kellemetlen órákat szerezett az átjáró szerencsétlen rendszergazdájának.

Mint mondottam, ez csak eddig jelentett gondot, mert a jelenlegi rendszermagok az `rp_filter` tűzfalszabályt használják e nehézség kiküszöbölésére. Ha az `rp_filter`-t bekapcsoljuk, minden olyan

csomagot eldob, amelyben a forráscím nem egyezik meg az útválasztó tábla megfelelő bejegyzésével. Ez ténylegesen megakadályozza az IP-cím hamisítását (spoofing), ugyanakkor lehetővé teszi az egyszerű és biztonságos álcázást, mint a fenti példánál.

Abban a kevéssé valószínű esetben, ha az `rp_filter` mégis gondot okoz, könnyen kikapcsolhatjuk:

```
# echo "0" > /proc/sys/net/ipv4/conf/all/rp_filter
```

### Kapcsolódó anyag

- [#47] IP Tables tippek és trükkök

## #47 IP Tables tippek és trükkök

### Hogyan lehet a tűzfalat az IP Tables programmal végzett egyszerű csomagszűrésnél több munkára fogni

Az `iptables` a Netfilter projekt számára készült tűzfalprogramok új nemzedékét képviseli. Az `iptables` elődjének valamennyi szolgáltatását, sőt még az állapottartó (stateful) tűzfalkezelést is támogatja. Az `iptables` keretrendszerként is működik, melynek segítségével betölthető modulokkal lehet képességeit bővíteni. A következőkben néhány olyan trükköt ismertetünk, amelyek az `iptables` alapterjesztésével használhatók, illetve néhány olyat is bemutatunk, amelyek az `iptables` számára készített bővítő modulokkal együtt használhatók.

E példáknál feltételezni fogjuk, hogy az alábbi környezeti változók beállítása már megtörtént:

`$EXT_IFACE` – a tűzfal külső, nyilvános hozzáférésű csatoló felülete.
`$INT_IFACE` – a tűzfal belső, magán hozzáférésű csatoló felülete.
`$DEST_IP` – az adott csomagnak a legvégső célcíme.

Az `iptables`-t használhatjuk az új bejövő csomagok korlátozására, hogy megakadályozzuk a szolgáltatásmegtagadás típusú támadásokat (Denial of Service attack, DoS). Ezt az alábbi szabályokkal érhetjük el:

```
# Create syn-flood chain for detecting Denial of
  ➥ Service attacks
iptables -t nat -N syn-flood

# Limit 12 connections per second (burst to 24)
iptables -t nat -A syn-flood -m limit
➥ --limit 12/s --limit-burst 24 -j RETURN
iptables -t nat -A syn-flood -j DROP

# Check for DoS attack
iptables -t nat -A PREROUTING -i $EXT_IFACE
➥ -d $DEST_IP -p tcp --syn -j syn-flood
```

Ezek a szabályok az új, bejövő TCP-kapcsolatok (olyan csomagok, amelyekben a SYN bit be van állítva) gyakoriságát 12/s-ra korlátozzák, amennyiben előzőleg a gyakoriság elérte a 24/s-os határt.

Az iptables használatával egy átlátszó (transparent) Squid proxykiszolgáló is telepíthető. Ez gyorstárként működik és naplóz minden kifelé, azaz internetre irányuló HTTP-kérést. A felhasználói böngésző programban nem igényel semmilyen módosítást, ugyanakkor hasznos a nem kívánt tartalom kitiltásában. Ezt a PREROUTING-lánc tetején elhelyezett iptables szabály segít elérnünk.

```
# Átlátszó Squid proxy a belső hálózat számára
#
# A Squid telepítéséről az alábbi
  ➥ címen olvashatunk:
# http://www.linuxdoc.org/HOWTO/mini/
  ➥ TransparentProxy.html
#
iptables -t nat -A PREROUTING -i $INT_IFACE
➥ -p tcp --dport 80 -j REDIRECT --to-port 3128
```

Ez a szabály a TCP 80-as kapura érkező kimenő kéréseket átirányítja a kiszolgálón futó Squid proxyhoz, amely a tűzfalon a 3128-as TCP-kaput használja.

Tetszőleges TCP-zászlók figyeltethetők az iptables-szel. Ez azt jelenti, hogy teljes „karácsonyfa"-csomagokat (az összes zászló bekapcsolva) és NULL csomagokat is blokkolhatunk az alábbi szabályokkal:

```
# DROP XMAS & NULL TCP csomagok
iptables -t nat -A PREROUTING -p tcp
➥ --tcp-flags ALL ALL -j DROP
iptables -t nat -A PREROUTING -p tcp
➥ --tcp-flags ALL NONE -j DROP
```

### Haladó szintű IP Tables tulajdonságok

Az iptables több haladó szintű tűzfalszolgáltatást vezetett be, amelyek a rendszermag foltozásával válnak elérhetővé. Ezek a foltok a http://www.netfilter.org címről szerezhetők be, az általunk használt iptables-nek megfelelő *patch-o-matic*-változat letöltésével. A *patch-o-matic* olyan iptables javítófoltok összessége, amelyek még nem kerültek be az általánosan használt Linux-rendszermagba (mainstream). Némelyik folt még csak próbaváltozat, ezért nagy elővigyázatossággal kell használni.

A Netfilter kísérleti jellegű psd foltjának alkalmazásával az iptables a következő szabály segítségével képes érzékelni és megakadályozni a kapupásztázásokat:

```
# A bejövo kapupásztázó csomagok eldobása
iptables -t nat -A PREROUTING -i $EXT_IFACE
➥ -d $DEST_IP -m psd -j DROP
```

A Netfilter *iplimit* nevű próbafoltja révén az iptables a következő szabállyal képes korlátozni az egy adott IP-címről létesített kapcsolatok számát:

```
# A több mint 16 aktív kapcsolatot fennetartó gépek
  ➥ csomagjaninak eldobása
iptables -t nat -A PREROUTING -i $EXT_IFACE
➥ -p tcp --syn -d $DEST_IP -m iplimit
➥ --iplimit-above 16 -j DROP
```

Az egyik leghatékonyabb Netfilter-folt lehetővé teszi, hogy csomagokat illesszünk össze a tartalmuk alapján. A láncillesztő próbafolt lehetővé teszi a bizonyos mintát követő foltok kiszűrését. Ez jól jöhet a CodeRed vagy Nimda vírusok kiszűrésénél, még mielőtt elérnék a webkiszolgálónkat. Ezt a következő szabállyal lehet elérni:

```
# CodeRed és Nimda virusokra utaló HTTP-csomagok
  ➥ eldobása
iptables -t filter -A INPUT -i $EXT_IFACE
➥ -p tcp -d $DEST_IP --dport http \
  -m string --string "/default.ida?" -j DROP
iptables -t filter -A INPUT -i $EXT_IFACE
➥ -p tcp -d $DEST_IP --dport http \
  -m string --string ".exe?/c+dir" -j DROP
iptables -t filter -A INPUT -i $EXT_IFACE
➥ -p tcp -d $DEST_IP --dport http \
  -m string --string ".exe?/c+tftp" -j DROP
```

Mostanra a kaputovábbítás (port forward) teljesen beépült az `iptables`-be (native). Ennek megvalósítására a NAT-tábla a `PREROUTING`-láncban egy Destination NAT (célcím, NAT) nevű szolgáltatást használ. A következő parancs a HTTP-kéréseket egy, a helyi hálózaton lévő rendszerre (10.0.0.3) továbbítja (port forward):

```
# http kaputovábbításra használjunk DNAT-ot.
iptables -t nat -A PREROUTING ! -i $INT_IFACE
➥ -p tcp --destination-port \
  80 -j DNAT --to 10.0.0.3:80
```

UDP-csomagok továbbítására is lehetőség nyílik. Ha az adatforgalmat egy bizonyos kapura továbbítjuk, akkor az `INPUT`-láncban szükségtelen olyan szabály, amely a bejövő kapcsolatokat fogadná ezen a kapun. Ez csak akkor fog működni, ha a cél a hálózaton egy helyileg csatlakoztatott csatolófelület, vagyis nem idegen hálózatokon levő célcímekre kell gondolni. Vessünk egy pillantást az olyan eszközökre, mint a `rinetd` ([#48] TCP kaputovábbítás tetszőleges számítógépekre) vagy az `nportredird`, ha az adatforgalmat távoli hálózatokra kell továbbítania.

Ha a HTTP-kéréseket egy, a belső hálózatra kapcsolt gépre továbbítjuk, akkor a CodeRed-vírust a `FORWARD`-láncból a következő paranccsal kiszűrhetjük:

```
iptables -t filter -A FORWARD -p tcp --dport http \
  -m string --string "/default.ida?" -j DROP
```

Az `iptables` használata első nekifutásra nagy kihívás lehet, de rugalmassága mérhetetlenül hasznos eszközzé teszi. Valahányszor elakadnánk szabálykészletünk kialakításában – mert egészen biztosan el fogunk akadni – mindig jusson eszünkbe a két jó barát az `iptables -L -n` és a `tcpdump` (amit esetleg egy gyors `ethereal` követhet).

### Kapcsolódó anyagok

- [#48] TCP kaputovábbítás tetszőleges számítógépekre
- The Linux Firewall HOWTO

## #48 TCP kaputovábbítás tetszőleges számítógépekre

### A nem helyi szolgáltatások helyi kapuról érkezőnek látszanak

Amint azt a [#47] „IP Tables tippek és trükkök" című részben láthattuk, az `iptables` segítségével egyszerűen lehet továbbítani TCP- és UDP-csomagokat a tűzfalról a helyi hálózat gépeire. De mi a helyzet akkor, ha az adatforgalmat egy tetszőleges címről kell továbbítani egy olyan gépre, ami még csak nem is a mi hálózatunkon van? Próbáljunk ki egy alkalmazási rétegben futó kaputovábbítót (application layer port forwarder), például a `rinetd`-t. (Megjegyzem, ha nem megbízható forrásból származik a program, célszerű a szerző GPG-kulcsával ellenőrizni a program hitelességét – egészséges üldözési mánia esetén mindent – Lektor.)

Ez a kicsinyke program csak néhány éves múltra tekint vissza, ám kis mérete és hatékony működése éppen az effajta feladatokhoz kínál megoldást. Csomagoljuk ki a tömörített állományt, és adjuk ki a `make` parancsot. Eredményül megkapjuk az apró `rinetd` végrehajtható bináris állományt, amely megengedi a TCP-kapuk tetszés szerinti címre történő továbbítását. Az UDP-kapukat a `rinetd` sajnos nem támogatja.

A beállításfájl nagyon egyszerű:

```
[Source Address] [Source Port]
➥ [Destination Address] [Destination Port]
```

Minden egyes továbbítandó kaput külön sorban kell feltüntetni. A forrás- és célcímek mind név szerint, mind IP-cím szerint megadhatók, és a 0.0.0.0-s IP-cím a `rinetd`-t minden helyben hozzáférhető IP-címhez hozzárendeli:

```
0.0.0.0 80 some.othersite.gov 80
216.218.203.211 25 123.45.67.89 25
0.0.0.0 5353 my.shellserver.us 22
```

Az állományt mentsük a */etc/rinetd.conf* nevű fájlba, és másoljuk valamilyen számunkra megfelelő helyre, mondjuk a */usr/local/sbin* könyvtárba. Ezután pedig a `rinetd` parancs kiadásával futtassuk a programot. Az első példa az összes helyi címnek szánt webfelületes adatforgalmat az *egymasik.webhely.hu* címre továbbítja. Meg kell jegyezni, hogy ez a módszer csak abban az esetben működik, ha nincs másik folyamat – mint például az Apache webkiszolgáló –, amely már korábban a helyi 80-as kapuhoz kapcsolódott.

A példa következő sora a 216.218.203.211 címre irányuló bejövő SMTP-forgalmat a 123.45.67.89 című levélkiszolgálóhoz továbbítja, de semmilyen egyéb helyi címhez kapcsolt SMTP-ügynök munkáját nem zavarja meg. A példa utolsó sora az 5353 kapun érkező bármilyen bejövő forgalmat a *shell.kiszolgalom.hu* gépen futó SSH kiszolgálónak fog továbbítani.

Mindez működik a NAT vagy bármilyen különleges rendszermag-beállítás nélkül, egyszerűen futtassuk az `rinetd`-t, amely démonként figyeli azokat a kapukat, amelyeket kijelöltünk számára.

Ez a segédprogram valóban megkönnyíti az áttérést a kiszolgálók újraszámozásánál vagy fizikai újra elhelyezésénél, minthogy úgy tűnhet, hogy az egyes szolgáltatások gyakran az eredeti IP-címen maradtak annak ellenére, hogy teljesen más hálózatról érkeznek.

A `rinetd`-nek nem szükséges rendszergazdai jogosultságokkal futnia, ha csak 1024-nél nagyobb kapukhoz kívánunk kapcsolódni. A hozzáférési jogosultságok szabályozásához és a naplózáshoz bőséges a kapcsolók kínálata. Ezt az aprócska eszközt megéri a kezünk ügyében tartani, amikor közvetett TCP-hívás zajlik.

### Kapcsolódó anyagok

- `http://www.boutell.com/rinetd/`
- [#47] IP Tables tippek és trükkök

## #49 Felhasználói láncok használata az IP Tablesben

### Tartsuk kézben a tűzfalat saját láncokkal

Az előre beállított értékeknek megfelelően az `iptables` szűrőtáblázat három láncból áll, ezek az `INPUT`, `FORWARD` és `OUTPUT`. Ezekhez további tetszőleges számú felhasználói lánc adható, hogy ilyen módon egyszerűsítsük a nagy szabálykészletek kezelését. A felhasználói láncok pontosan úgy viselkednek, mint a beépített láncok, bevezetve a logikát, amelyet tovább kell adni, mielőtt a csomag végső sorsa felől határoznánk.

Új lánc létrehozásához használjuk a `-N` kapcsolót:

```
root@mouse:~# iptables -N fun-filter
```

A szabályos `-L` kapcsolóval bármikor megtekinthetjük, milyen láncok vannak kijelölve:

```
root@mouse:~# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination
```

```
Chain fun-filter (0 references)
target prot opt source destination
```

Annak érdekében, hogy a frissen létrehozott felhasználói láncot használni tudjuk, valahonnan be kell lépni rá. Adjunk az INPUT-lánchoz egy, a frissen létrehozott fun-filter láncra mutató ugrást:

```
root@mouse:~# iptables -t filter -A INPUT
➥ -j fun-filter
```

A felhasználói láncot tetszés szerinti bonyolultságig lehet fűzni. Például, ha szeretnénk az adatcsomagokat a forrás-MAC-címek alapján összeilleszteni, az alábbi láncokat kell létrehozni:

```
root@mouse:~# iptables -A fun-filter -m mac
➥ --mac-source 11:22:33:aa:bb:cc -j ACCEPT
root@mouse:~# iptables -A fun-filter -m mac
➥ --mac-source de:ad:be:ef:00:42 -j ACCEPT
root@mouse:~# iptables -A fun-filter -m mac
➥ --mac-source 00:22:44:fa:ca:de
➥ -j REJECT --reject-with icmp-host-unreachable
root@mouse:~# iptables -A fun-filter -j RETURN
```

A tábla végén levő RETURN ugrás arra utasítja a feldolgozást, hogy a láncban ugyanoda lépjen vissza, ahonnan ide került, jelen esetben ez nem más, mint az INPUT-lánc. A -L kapcsolóval ismételten megnézhetjük, hogyan festenek tábláink:

```
root@mouse:~# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
fun-filter all -- anywhere anywhere

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Chain fun-filter (0 references)
target prot opt source destination
```

```
ACCEPT all -- anywhere anywhere MAC
➥ 11:22:33:AA:BB:CC
ACCEPT all -- anywhere anywhere MAC
➥ DE:AD:BE:EF:00:42
REJECT all -- anywhere anywhere MAC
➥ 00:22:44:FA:CA:DE
➥ reject-with icmp-host-unreachable
RETURN all -- anywhere anywhere
```

Tetszőleges számú, felhasználó által meghatározott láncot lehet alkalmazni, akár egyikről a másikra ugrálva is. Ez segít elkülöníteni az éppen kialakított szabályokat a szabványos rendszerházirend szabályok közül, és könnyedén ki, illetve be lehet kapcsolni őket. Ha ideiglenesen le szeretnénk állítani a felhasználói láncot, akkor egyszerűen töröljük az ugrást az `INPUT`-láncból:

```
root@mouse:~# iptables -t filter -D INPUT
➥ -j fun-filter
```

Ha a felhasználói láncot akarjuk törölni, akkor használjuk a `-X` kapcsolót:

```
root@mouse:~# iptables -X fun-filter
```

Meg kell említeni, hogyha megpróbáljuk törölni a felhasználói láncot, akkor már nem lehet rá hivatkozni, ezért célszerű először kiüríteni a `-F` kapcsolóval, ha még akadnak rá hivatkozó szabályok.

Egy megfelelően karbantartott `iptables` segítségével még a legösszetettebb szabálykészletek is könnyedén olvashatók, amennyiben beszédes nevekkel ellátott felhasználói láncokat használunk.

### Kapcsolódó anyag

- [#47] IP Tables tippek és trükkök

# #50 Alagutazás: IP-IP tokozás

## IP-alagutazás a Linux IP-IP-vel

Ha még sohasem használtunk IP-alagutazást, akkor a folytatás előtt érdemes egy pillantást vetnünk az Advanced Router (Fejlett útválasztó című) Linux HOGYAN-ra. Az IP-alagút lényegében nagyon hasonlít a VPN-re, azzal a különbséggel, hogy nem minden alagút titkosított. (Fontos megjegyezni az `ipsec` programot, mellyel könyedén és biztonságosan alakíthatunk ki két hálózatrész között VPN-kapcsolatot. Sőt, ma már a legtöbb útválasztó DSL-eszköz is támogatja ezt a megoldást – Lektor.)

A gépnek, ami egy másik hálózatba alagúton keresztül csatlakozik, van egy virtuális csatolófelülete, amely nem a helyi, hanem a távoli hálózaton érvényes IP-címmel van ellátva. Rendszerint a teljes hálózati forgalmat vagy annak legnagyobb részét az útválasztó ezen az alagúton tereli keresztül olyan módon, hogy a távoli ügyfelek feltűnnek a hálózati szolgáltatások között, vagy másképp fogalmazva: két magánhálózatot az internet segítségével lehet összekapcsolni, hogy az bonyolítsa le az adatforgalmat.

Ha két gép között egyszerű IP az IP-n belüli alagutazást akarunk megvalósítani, akkor biztosan jó lenne kipróbálni az IP-IP-t. Jelenleg valószínűleg ez az elérhető legegyszerűbb alagútprotokoll, amely képes együttműködni a BSD, Solaris és Windows operációs rendszerrel. De ne feledkezzünk meg róla, hogy az IP-IP csupán egy alagutazó protokoll, amelybe semmiféle titkosítás nincs beépítve.

Ez csupán egycímes (unicast) csomagok alagutazására képes; amennyiben csoportszórásos adatforgalomra van szükség, akkor a GRE-alagutazást kell tanulmányoznunk.

Mielőtt berobognánk az első alagútba, szükség lesz az Advanced Router eszközök, név szerint az IP segédprogram, másolatára. A legfrissebb hiteles példányt az `ftp://ftp.inr.ac.ru/ip-routing` címről lehet beszerezni. Szeretnénk felhívni a figyelmet arra, hogy az Advanced Router eszközök nem különösebben barátságosak, de lehetővé teszik, hogy a linuxos hálózati eszköz szinte minden oldalát kihasználjuk.

Tételezzük fel, hogy két magánhálózatot (10.42.1.0/24 és 10.42.2.0/24) működtetünk, és mindkettő egy-egy, az adott hálózatban levő linuxos útválasztón keresztül kapcsolódik az internethez. Az első hálózati útválasztó „valódi" címe 240.101.83.2, miközben a második útválasztó „valódi" IP-címe 251.4.92.217. Legelőször mindkét útválasztón töltsük be a szükséges rendszermagmodult:

```
# modprobe ipip
```

A következő műveletsort az első hálózati útválasztón (10.42.1.0/24) végezzük el:

```
# ip tunnel add mytun mode ipip remote 251.4.92.217 \
  local 240.101.83.2 ttl 255
# ifconfig mytun 10.42.1.1
# route add -net 10.42.2.0/24 dev mytun
```

A válaszlépéseket pedig a második útválasztón (10.42.2.0/24) hajtjuk végre:

```
# ip tunnel add mytun mode ipip remote 240.101.83.2 \
  local 251.4.92.217 ttl 255
# ifconfig tun10 10.42.2.1
# route add -net 10.42.1.0/24 dev mytun
```

Természetesen a csatolófelületnek a mytun névnél beszédesebbet is ki lehet találni. Ha minden rendben, akkor az első hálózati útválasztóról pingelhető a 10.42.2.1-es gép, ugyanakkor a másik útválasztóról pingelhető a 10.42.1.1-es gép.

Hasonlóképpen a 10.42.1.0/24 hálózatra kapcsolódó összes gépnek az útválasztón keresztül kapcsolatba kell tudnia lépni a 10.42.2.0/24-es hálózatra csatlakozó összes géppel, pontosan úgy, mintha közvetlenül lenne összekötve a két hálózat.

Ha Linux-rendszermagunk változatszáma 2.2x szerencsénk van: létezik egyszerűbb megoldás is, amellyel az Advanced Router eszközcsomag teljes mértékben kiváltható. A rendszermagmodul betöltése után inkább a következő parancsokat próbáljuk alkalmazni:

```
# ifconfig tun10 10.42.1.1 pointopoint 251.4.92.217
# route add -net 10.42.2.0/24 dev tun10
```

A második hálózati útválasztón pedig a következőket hajtsuk végre:

```
# ifconfig tun10 10.42.2.1 pointopoint 240.101.83.2
# route add -net 10.42.1.0/24 dev tun10
```

Ha meg tudjuk pingelni az ellenoldali útválasztót, de a hálózat többi gépe nem tud csomagot küldeni az útválasztó mögé, győződjünk meg róla, hogy mindkét útválasztón be van-e állítva, hogy a csomagokat a csatolók között továbbítsa:

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
```

Amennyiben a 10.42.1.0 és a 10.42.2.0 hálózatokon kívül továbbiakat is szeretnénk összekapcsolni, akkor ezeket a `route add -net...` sorokkal vehetjük fel a listába. A hálózatra kapcsolt gépeken semmiféle beállításra nincs szükség, amennyiben már be van állítva számukra az alapértelmezett átjáró (ami minden bizonnyal így is van, hiszen végső soron ez a saját útválasztójuk).

Az alagút leállításához mindkét útválasztón le kell állítani a csatolófelületet, s ha nincs már rájuk szükség, akár törölhetjük is őket:

```
# ifconfig mytun down
# ip tunnel del mytun
```

vagy a Linux 2.2-es rendszermagban:

```
# ifconfig tun10 down
```

Amikor a hálózati csatolófelület megszűnik, a rendszermag udvariasan eltakarítja az útválasztó táblát.

### Kapcsolódó anyagok

- Haladó útválasztó HOGYAN:
  `http://www.tldp.org/HOWTO/Adv-Routing-HOWTO/`
- Haladó útválasztó eszközök (`iproute2`):
  `ftp://ftp.inr.ac.ru/ip-routing`

# Alagutazás: GRE-tokozás

## IP-alagút általános útválasztó tokozással

A GRE rövidítés a Generic Routing Encapsulation (általános útválasztó tokozás) elnevezést takarja. Az IP-IP-alagutazáshoz (#50 Alagutazás: IP-IP tokozás) hasonlóan a GRE-tokozás is titkosítás nélküli protokoll. A GRE előnye az IP-IP tokozással szemben az, hogy támogatja a csoportos üzenetszórásos csomagok használatát (multicast packet) és a CISCO útválasztókkal is képes együttműködni.

Ezúttal is feltételezzük két hálózat meglétét (10.42.1.0/24 és 10.42.2.0/24), és azt, hogy mindkét hálózat egy-egy linuxos útválasztón keresztül közvetlenül kapcsolódik az internethez. Az első hálózatra kapcsolt útválasztó „tényleges" címe 240.101.83.2, ezzel szemben a másik, hálózatra kapcsolt útválasztó „valódi" címe 251.4.92.217. Amint az IP-IP alagutazásnál (#50 Alagutazás: IP-IP tokozás) ezúttal is szükség lesz a haladó szintű útválasztó csomag egy példányára (a Linux 2.2-esben nem tudtam olyan megoldást találni, amivel a GRE-alagutat egyszerűbben lehetett volna alkalmazni). Az *iproute2* csomag telepítése után a teendőinket mindkét útválasztón a GRE rendszermagmodulok betöltésével kezdjük:

```
# modprobe ip_gre
```

Az első hálózati útválasztón hozzunk létre egy új hálózati útválasztó készüléket:

```
# ip tunnel add gre0 mode gre remote 251.4.92.217
➥ clocal 240.101.83.2 ttl 255
# ip addr add 10.42.1.254 dev gre0
# ip link set gre0 up
```

Megjegyezzük, hogy a csatolót tetszés szerint el lehet nevezni, a `gre0` csupán egy példa. Továbbá a 10.42.1.254 cím az első hálózaton belül bármilyen szabad cím lehet, kivéve a 10.42.1.1-et, hiszen ez a belső csatolófelülethez kapcsolódik.

Ezt követően a hálózati útválasztókra vegyük fel a hálózati útvonalakat az új alagútútválasztókon keresztül:

```
# ip route add 10.42.2.0/24 dev gre0
```

Ezzel kész is az első hálózat, következzen most a második:

```
# ip tunnel add gre0 mode gre remote 240.101.83.2
➥ local 251.4.92.217 ttl 255
# ip addr add 10.42.2.254 dev gre0
# ip link set gre0 up
# ip route add 10.42.1.0/24 dev gre0
```

A 10.42.2.254-es IP-cím bármelyik szabad cím lehet a második hálózaton. Az útválasztó beállításait egészítsük ki annyi `ip route add ... dev gre0` paranccsal, amennyi szükséges.

Ezennel a mű el is készült: most már úgy tudunk a két hálózat között csomagokat küldeni, mintha közvetlenül lennének összekötve az internet helyett. Az első hálózaton elindított `traceroute` néhány ugrással a második hálózat bármelyik gépét eléri – de minden bizonnyal fel fog tűnni egy kis várakozási idő, amikor a 10.42.2.254-es ugrás történik, kivéve a valóban kifogástalan kapcsolatot. Ha mégsem járnánk sikerrel erőfeszítéseink, vessünk egy pillantást az IP-IP-példánál leírtakra. Új hálózat beállításakor a legmegbízhatóbb barátok a `tcpdump` és az `ethereal` csomagvizsgálók (packet sniffer). A `tcpdump 'proto \icmp'` parancs egyidejű futtatása a két kiszolgálón pingelés közben nagyon részletes áttekintést ad arról, mi is történik valójában.

Az alagút leválasztásához futtassuk ezeket a parancsokat mindkét útválasztón:

```
# ip link set gre0 down
# ip tunnel del gre0
```

# #52 A vtun használata az SSH felett a NAT megkerülésére

## Két hálózat összekapcsolása a vtun használatával és egyetlen SSH-kapcsolattal

A vtun felhasználói térben működő, a tun általános rendszermagbeli meghajtót használó kiszolgálóprogram, amely lehetővé teszi teljes hálózatok számára, hogy egymásba vezető alagutat építsenek. A kapcsolat létrejöhet közvetlenül az IP felett, a PPP-n vagy a soros kapun keresztül. Ez a módszer akkor lehet különösen hasznos, ha az általános hálózati hozzáférést egy köztes tűzfal korlátozza, minthogy a teljes hálózati forgalom titkosítva kerül továbbításra egyetlen TCP-kapun keresztül (egészen pontosan egy SSH-kapcsolaton keresztül).

Az alábbiakban ismertetett eljárás egy magánhálózati címmel (10.42.4.6) ellátott gépen új alagút-csatolófelületet éleszt fel, valódi, élő útválasztón keresztül küldött (routed) címmel (208.201.239.33), amely a várakozásoknak megfelelően működik, úgy mintha a magánhálózat ott sem lenne. Ezt úgy érjük el, hogy felépítjük az alagutat, eldobjuk az eredeti útválasztóbeállítást, majd egy új alapértelmezett, az alagút másik végén keresztül vezető útválasztást adunk hozzá. Elsőként a hálózat alagutazás előtti beállítását láthatjuk:

```
root@client:~# ifconfig eth2
eth2 Link encap:Ethernet HWaddr 00:02:2D:2A:27:EA
inet addr:10.42.3.2 Bcast:10.42.3.63
➥ Mask:255.255.255.192
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:662 errors:0 dropped:0
➥ overruns:0 frame:0
TX packets:733 errors:0 dropped:0
➥ overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:105616 (103.1 Kb) TX bytes:74259 (72.5 Kb)
Interrupt:3 Base address:0x100

root@client:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric
➥ Ref Use Iface
```

```
10.42.3.0 * 255.255.255.192 U 0 0 0 eth2
loopback * 255.0.0.0 U 0 0 0 lo
default 10.42.3.1 0.0.0.0 UG 0 0 0 eth2
```

Amint az látható, a helyi hálózatunk címe 10.42.3.0/26, IP-címünk 10.42.3.2 és az alapértelmezés szerinti átjárónk 10.42.3.1. Ez az átjáró hálózati címfordítást (Network Address Translation, NAT) biztosít az internet számára. Íme, így néz ki a *yahoo.com* elérési útvonala:

```
root@client:~# traceroute -n yahoo.com
traceroute to yahoo.com (64.58.79.230),
➥ 30 hops max, 40 byte packets
1 10.42.3.1 2.848 ms 2.304 ms 2.915 ms
2 209.204.179.1 16.654 ms 16.052 ms 19.224 ms
3 208.201.224.194 20.112 ms 20.863 ms 18.238 ms
4 208.201.224.5 213.466 ms 338.259 ms 357.7 ms
5 206.24.221.217 20.743 ms 23.504 ms 24.192 ms
6 206.24.210.62 22.379 ms 30.948 ms 54.475 ms
7 206.24.226.104 94.263 ms 94.192 ms 91.825 ms
8 206.24.238.61 97.107 ms 91.005 ms 91.133 ms
9 206.24.238.26 95.443 ms 98.846 ms 100.055 ms
10 216.109.66.7 92.133 ms 97.419 ms 94.22 ms
11 216.33.98.19 99.491 ms 94.661 ms 100.002 ms
12 216.35.210.126 97.945 ms 93.608 ms 95.347 ms
13 64.58.77.41 98.607 ms 99.588 ms 97.816 ms
```

Példánkban egy alagútkiszolgálót kapcsolunk az internetre a 208.201.239.5-ös címen. Ennek két tartalék élő IP-címe van (208.201.239.32 és 208.201.239.33) az alagúthoz. Erre a gépre kiszolgálóként fogunk hivatkozni, a helyi gépünkre pedig ügyfélként.

Most indítsuk be az alagutat. Ehhez mindkét gépen töltsük be a *tun* meghajtóprogramot:

```
# modprobe tun
```

Előfordulhat, hogy a *tun* nem működik, ha a kiszolgáló- és ügyféloldali rendszermag-változatok nem egyeznek meg. A legjobb eredmény elérése érdekében mindkét gépen használjuk ugyanazt a rendszermagot, lehetőleg a legfrissebbet.

A kiszolgálógépen /usr/local/etc/vtund.conf néven hozzuk létre a következő állományt:

```
options {
port 5000;
ifconfig /sbin/ifconfig;
route /sbin/route;
syslog auth;
}

default {
compress no;
speed 0;
}

home {
type tun;
proto tcp;
stat yes;
keepalive yes;

pass sHHH; # Password is REQUIRED.

up {
ifconfig "%% 208.201.239.32 pointopoint
208.201.239.33";

program /sbin/arp "-Ds 208.201.239.33 %% pub";
program /sbin/arp "-Ds 208.201.239.33 eth0 pub";

route "add -net 10.42.0.0/16 gw 208.201.239.33";
} ;

down {
program /sbin/arp "-d 208.201.239.33 -i %%";
program /sbin/arp "-d 208.201.239.33 -i eth0";

route "del -net 10.42.0.0/16 gw 208.201.239.33";
} ;
}
```

Indítsuk el a vtund kiszolgálót az alábbi paranccsal:

```
root@server:~# vtund -s
```

Most az ügyféloldalon is hozzuk létre a */usr/local/etc/vtund.conf* állományt a következő tartalommal:

```
options {
port 5000;
ifconfig /sbin/ifconfig;
route /sbin/route;
}

default {
compress no;
speed 0;
}

home {
type tun;
proto tcp;
keepalive yes;

pass sHHH; # Password is REQUIRED.

up {
ifconfig "%% 208.201.239.33
➥ pointopoint 208.201.239.32 arp";

route "add 208.201.239.5 gw 10.42.3.1";
route "del default";
route "add default gw 208.201.239.32";

} ;

down {
route "del default";
route "del 208.201.239.5 gw 10.42.3.1";
route "add default gw 10.42.3.1";
} ;
}
```

Végül hajtsuk végre ezt a parancsot az ügyféloldalon:

```
root@client:~# vtund -p home server
```

A mű elkészült! Nemcsak felépítettünk egy alagutat a kiszolgálógép és az ügyfélgép között, hanem az alagút másik végén keresztül új alapértelmezett útválasztást is meghatároztunk. Lássuk csak mi is történik, ha újra kiadjuk a `traceroute` parancsot a *yahoo.com* címre, miközben alagutunk a helyén van:

```
root@client:~# traceroute -n yahoo.com
traceroute to yahoo.com (64.58.79.230),
➥ 30 hops max, 40 byte packets
1 208.201.239.32 24.368 ms 28.019 ms 19.114 ms
2 208.201.239.1 21.677 ms 22.644 ms 23.489 ms
3 208.201.224.194 20.41 ms 22.997 ms 23.788 ms
4 208.201.224.5 26.496 ms 23.8 ms 25.752 ms
5 206.24.221.217 26.174 ms 28.077 ms 26.344 ms
6 206.24.210.62 26.484 ms 27.851 ms 25.015 ms
7 206.24.226.103 104.22 ms 114.278 ms 108.575 ms
8 206.24.238.57 99.978 ms 99.028 ms 100.976 ms
9 206.24.238.26 103.749 ms 101.416 ms 101.09 ms
10 216.109.66.132 102.426 ms 104.222 ms 98.675 ms
11 216.33.98.19 99.985 ms 99.618 ms 103.827 ms
12 216.35.210.126 104.075 ms 103.247 ms 106.398 ms
13 64.58.77.41 107.219 ms 106.285 ms 101.169 ms
```

Ez azt jelenti, hogy az ügyfélen futó bármelyik kiszolgáló-folyamat most már hozzáférhető az internet számára a 208.201.239.33-as IP-címen. Mindez olyan módon, hogy a 10.42.3.1 átjárón semmiféle módosítást nem kellett végezni (például kaputovábbítás).

Az alagút csatolófelülete az ügyféloldalon így fest:

```
root@client:~# ifconfig tun0
tun0 Link encap:Point-to-Point Protocol
inet addr:208.201.239.33 P-t-P:208.201.239.32
➥ Mask:255.255.255.255
UP POINTOPOINT RUNNING MULTICAST MTU:1500 Metric:1
RX packets:39 errors:0 dropped:0 overruns:0 frame:0
```

```
TX packets:39 errors:0 dropped:0
➥ overruns:0 carrier:0
collisions:0 txqueuelen:10
RX bytes:2220 (2.1 Kb) TX bytes:1560 (1.5 Kb)
```

A következő kódrészlet pedig a frissített útválasztó táblát mutatja. Vegyük figyelembe, hogy továbbra is szükségünk van a régi alapértelmezett átjárónkon keresztül az alagútkiszolgáló IP-címéhez vezető gépelérési útvonalra (host route), különben az alagút forgalma nem tud kijutni:

```
root@client:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric
➥ Ref Use Iface
208.201.239.5 10.42.3.1 255.255.255.255
➥ UGH 0 0 0 eth2
208.201.239.32 * 255.255.255.255 UH 0 0 0 tun0
10.42.3.0 * 255.255.255.192 U 0 0 0 eth2
10.42.4.0 * 255.255.255.192 U 0 0 0 eth0
loopback * 255.0.0.0 U 0 0 0 lo
default 208.201.239.32 0.0.0.0 UG 0 0 0 tun0
```

Az alagút leállításához egyszerűen ki kell lőni az ügyféloldalon a vtund folyamatot. Ez vissza fogja állítani a különböző hálózati beállítást az eredeti állapotába.

Ez a módszer remekül működik, ha az ember rábízza magát a vtunra, hogy használjon erős titkosítást, és ne legyenek benne távolról kihasználható biztonsági rések. Szerintem senki sem lehet elég paranoiás, ha internethez csatlakozó géppel van dolga. A vtun SSH feletti használatához (és hogy támaszkodjunk az SSH nyújtotta erős hitelesítési és titkosítási módszerre) egyszerűen irányítsuk át az ügyfél 5000-es kapuját a kiszolgáló megegyező kapujára. Tegyünk egy próbát:

```
root@client:~# ssh -f -N -c blowfish
➥ -C -L5000:localhost:5000 server
root@client:~# vtund -p home localhost
root@client:~# traceroute -n yahoo.com
traceroute to yahoo.com (64.58.79.230),
➥ 30 hops max, 40 byte packets
```

```
1 208.201.239.32 24.715 ms 31.713 ms 29.519 ms
2 208.201.239.1 28.389 ms 36.247 ms 28.879 ms
3 208.201.224.194 48.777 ms 28.602 ms 44.024 ms
4 208.201.224.5 38.788 ms 35.608 ms 35.72 ms
5 206.24.221.217 37.729 ms 38.821 ms 43.489 ms
6 206.24.210.62 39.577 ms 43.784 ms 34.711 ms
7 206.24.226.103 110.761 ms 111.246 ms 117.15 ms
8 206.24.238.57 112.569 ms 113.2 ms 111.773 ms
9 206.24.238.26 111.466 ms 123.051 ms 118.58 ms
10 216.109.66.132 113.79 ms 119.143 ms 109.934 ms
11 216.33.98.19 111.948 ms 117.959 ms 122.269 ms
12 216.35.210.126 113.472 ms 111.129 ms 118.079 ms
13 64.58.77.41 110.923 ms 110.733 ms 115.22 ms
```

Ahhoz, hogy elutasítsuk a kiszolgáló 5000-es kapuján futó `vtund`-hez irányuló kapcsolatokat, hozzunk létre egy Netfilter-szabályt, amely eldobja a külvilág felől érkező kapcsolódási kéréseket:

```
root@server:~# iptables -A INPUT -t filter
➥ -i eth0 -p tcp --dport 5000 -j DROP
```

Ez a helyi kapcsolatokat átengedi (mivel ezek a hurokeszközt használják), ezért a kapcsolatok elfogadása előtt a kiszolgálóhoz SSH-alagútra van szükség.

Amint látható, ez nagyon hasznos eszköz. A NAT mögötti gépek élő IP-címekkel való ellátásán kívül gyakorlatilag összekapcsolhatunk bármely két hálózatot, ha létre tudunk hozni közöttük egy SSH-kapcsolatot, bármelyik végéről indítva is.

Ha már zsong a fejünk a fenti *vtund.conf* beállítástól, vagy ha végletekig lusták lennénk és nem is szándékoznánk a fejünket törni azon, hogy mit kell változtatni a példán a saját ügyféloldali *vtund.conf* állomány létrehozásához, akkor vessünk egy pillantást az önműködő *vtund.conf* létrehozó programra, lásd az [#53] Önműködő *vtund.conf* létrehozása.

### Megjegyzések

- A munkamenet (session) nevének (ez a fenti példában home) azonosnak kell lennie mind a kiszolgáló-, mind az ügyféloldalon, különben csak a kétértelmű „Kiszolgáló leválasztva" üzenetet kapjuk.
- Ugyanez vonatkozik mindkét oldalon a *vtund.conf* állományban a jelszómezőre. Jelen kell lennie, és mindkét oldalon azonosnak kell lennie, különben a kapcsolat nem fog működni.
- Ha a kapcsolódás sikertelen, győződjünk meg róla, hogy a kiszolgálógép üzemel-e, illetve ugyanazt azt a rendszermag-változatot használjuk-e mindkét oldalon. A kiszolgáló állapotát az ügyféloldalról a `telnet server 5000` paranccsal ellenőrizhetjük.
- Először a közvetlen eljárással próbálkozzunk, és csak akkor fogjuk munkára az SSH-t, ha már elégedettek vagyunk a *vtund.conf* beállításokkal.
- Ha továbbra is gondjaink merülnek fel, ellenőrizzük a */etc/syslog.conf* állományt, hogy megtudjuk hová kerülnek az `auth` üzenetek, és a kapcsolódási kísérlet során vizsgáljuk meg a naplót, mind az ügyfél-, mind a kiszolgálóoldalon.

### Kapcsolódó anyagok

- A `vtun` honlapja a `http://vtun.sourceforge.net/` érhető el.
- */usr/src/linux/Documentation/networking/tuntap.txt*
- `man vtund`, `man vtund.conf`
- [#53] Önműködő *vtund.conf* létrehozása
- [#45] Tűzfal létrehozása egy tetszőleges kiszolgáló parancssorából
- [#71] Kaputovábbítás SSH-n keresztül

## #53 Önműködő vtund.conf-létrehozó

### Vtund.conf létrehozása röptében, hogy illeszkedjen a változó hálózati feltételekhez

Amennyiben közvetlenül az [#51] „Alagutazás: GRE-tokozás" részből ugrottunk ide: ez a héjprogram egy működőképes *vtund.conf* állományt hoz létre önműködően az ügyféloldalon.

Ha viszont kimaradt volna az #51 trükk vagy még sohasem használtunk `vtun`-t, akkor térjünk vissza oda és olvassuk el, mielőtt megpróbálnánk kinyerni az erőt e morzsányi Perl-programból. Ez a programocska lényegében megkísérli találgatás nélkül megváltoztatni az útválasztó táblát az ügyféloldalon, önműködően érzékelve az alapértelmezett átjárót, s ennek megfelelően építi fel a *vtund.conf* állományt.

A héjprogram beállításához vessünk egy pillantást a beállító szakaszra. A `$Config` első sorában találjuk az #51 „Alagutazás: GRE-tokozás" trükkben is használt címeket, kapukat, valamint jelszavakat. A második sor egy egyszerű példa, hogyan adhatunk hozzá többet is.

A héjprogram futtatásához adjuk ki a `vtundconf home` parancsot, vagy a `$TunnelName` változó értékét állítsuk be alapértelmezettként a kívánt értékre. Még jobb, ha egy a programra mutató közvetett hivatkozást hozunk létre:

```
# ln -s vtundconf home
# ln -s vtundconf tunnel2
```

Azután a közvetett hivatkozás közvetlen meghívásával hozzuk létre a megfelelő *vtund.conf* állományt:

```
# vtundconf home > /usr/local/etc/vtund.conf
```

Vajon miért veszi bárki is a fáradságot, hogy elsőként a *vtund.conf* héjprogramot készítse el? Ha már egyszer valaki a helyes beállítások birtokába kerül, akkor azt a későbbiekben talán már sohasem kell megváltoztatni, igaz?

Valóban, ez rendszerint így van. Azonban tételezzük fel azt az esetet, hogy a linuxos gép a nap során sokféle hálózathoz kapcsolódik: otthon DSL-vonalhoz, az irodában ethernethálózathoz, végül a helyi kávézóban, mondjuk, vezeték nélküli hálózathoz. Mindegyik helyszínen a *vtund.conf* egyszeri futtatásával azonnal egy működőképes összeállításhoz juthatunk, még akkor is, ha az IP-címet és az átjárót DHCP-kiszolgáló határozza meg a gép számára. Ez egy „élő", routeolható címmel ellátott gép bekapcsolását és üzemeltetését nagyon megkönnyíti, tekintet nélkül a helyi hálózat elrendezésére.

Mellékesen azt is meg kell említeni, hogy a vtund és *vtund.conf* jelenleg nagyszerűen működtethetők Linux, FreeBSD, OS X, Solaris, és még néhány másik operációs rendszeren.

### Lista: vtundconf

```
#!/usr/bin/perl -w

# vtund wrapper in need of a better name.
#
# (c)2002 Schuyler Erle & Rob Flickenger
#
################ CONFIGURATION

# If TunnelName is blank, the wrapper will look at
  ➥ @ARGV or $0.
#
# Config is TunnelName, LocalIP, RemoteIP,
  ➥ TunnelHost, TunnelPort, Secret
#
my $TunnelName = "";
my $Config = q{
home 208.201.239.33 208.201.239.32
➥ 208.201.239.5 5000 sHHH
tunnel2 10.0.1.100 10.0.1.1 192.168.1.4 6001 foobar
} ;

########### MAIN PROGRAM BEGINS HERE

use POSIX 'tmpnam';
use IO::File;
use File::Basename;
use strict;

# Where to find things...
#
$ENV{ PATH} =
➥ "/bin:/usr/bin:/usr/local/bin:/sbin:/usr/
➥ sbin:/usr/local/sbin";
my $IP_Match = '((?:\ d{ 1,3} \ .){ 3} \ d{ 1,3} )';
➥ # match xxx.xxx.xxx.xxx
my $Ifconfig = "ifconfig -a";
```

```
my $Netstat = "netstat -rn";
my $Vtund = "/bin/echo";
my $Debug = 1;

# Load the template from the data section.
#
my $template = join( "", <DATA> );

# Open a temp file -- adapted from Perl Cookbook,
# 1st Ed., sec. 7.5.
#
my ( $file, $name ) = ("", "");
$name = tmpnam(  ) until $file =
➥ IO::File->new( $name, O_RDWR|O_CREAT|O_EXCL );
END {  unlink( $name ) or warn
➥ "Can't remove temporary file $name!\ n"; }

# If no TunnelName is specified, use the first
# thing on the command line,
# or if there isn't one, the basename
# of the script.
# This allows users to symlink different tunnel
# names to the same script.
#
$TunnelName ||= shift(@ARGV) || basename($0);
die "Can't determine tunnel config to use!\ n"
➥ unless $TunnelName;

# Parse config.
#
my ($LocalIP, $RemoteIP, $TunnelHost, $TunnelPort,
➥ $Secret);
for (split(/\ r*\ n+/, $Config)) {
my ($conf, @vars) =
➥ grep( $_ ne "", split( /\ s+/ ));
next if not $conf or $conf =~
➥ /^\ s*#/o; # skip blank lines, comments
if ($conf eq $TunnelName) {
($LocalIP, $RemoteIP, $TunnelHost, $TunnelPort,
➥ $Secret) = @vars;
last;
}
}
```

```
die "Can't determine configuration
➥ for TunnelName '$TunnelName'!\ n"
unless $RemoteIP and $TunnelHost and $TunnelPort;

# Find the default gateway.
#
my ( $GatewayIP, $ExternalDevice );

for (qx{  $Netstat } ) {
# In both Linux and BSD, the gateway is the next
# thing on the line,
# and the interface is the last.
#
if ( /^(?:0.0.0.0|default)\
➥ s+(\ S+)\ s+.*?(\ S+)\ s*$/o ) {
$GatewayIP = $1;
$ExternalDevice = $2;
last;
}
}

die "Can't determine default gateway!\ n"
➥ unless $GatewayIP and $ExternalDevice;

# Figure out the LocalIP and LocalNetwork.
#
my ( $LocalNetwork );
my ( $iface, $addr, $up, $network, $mask ) = "";

sub compute_netmask {
($addr, $mask) = @_;
# We have to mask $addr with $mask because linux
# /sbin/route
# complains if the network address doesn't match
# the netmask.
#
my @ip = split( /\ ./, $addr );
my @mask = split( /\ ./, $mask );
$ip[$_] = ($ip[$_] + 0) & ($mask[$_] + 0)
➥ for (0..$#ip);
$addr = join(".", @ip);
return $addr;
}
```

```
for (qx{  $Ifconfig } ) {
last unless defined $_;

# If we got a new device, stash the previous one
# (if any).
if ( /^([^\ s:]+)/o ) {
if ( $iface eq $ExternalDevice
➥ and $network and $up ) {
$LocalNetwork = $network;
last;
}
$iface = $1;
$up = 0;
}

# Get the network mask for the current interface.
if ( /addr:$IP_Match.*?mask:$IP_Match/io ) {
# Linux style ifconfig.
compute_netmask($1, $2);
$network = "$addr netmask $mask";
}  elsif ( /inet $IP_Match.*?mask
➥ 0x([a-f0-9]{ 8} )/io ) {
# BSD style ifconfig.
($addr, $mask) = ($1, $2);
$mask = join(".", map( hex $_, $mask =~
➥ /(..)/gs ));
compute_netmask($addr, $mask);
$network = "$addr/$mask";
}

# Ignore interfaces that are loopback devices
# or aren't up.
$iface = "" if /\ bLOOPBACK\ b/o;
$up++ if /\ bUP\ b/o;
}

die "Can't determine local IP address!\ n"
➥ unless $LocalIP and $LocalNetwork;

# Set OS dependent variables.
#
my ( $GW, $NET, $PTP );
```

```
if ( $^O eq "linux" ) {
$GW = "gw"; $PTP = "pointopoint"; $NET = "-net";
}  else {
$GW = $PTP = $NET = "";
}

# Parse the config template.
#
$template =~ s/(\ $\ w+)/$1/gee;

# Write the temp file and execute vtund.
#
if ($Debug) {
print $template;
}  else {
print $file $template;
close $file;
system("$Vtund $name");
}

_  _DATA_  _

options {
port $TunnelPort;
ifconfig /sbin/ifconfig;
route /sbin/route;
}

default {
compress no;
speed 0;
}

$TunnelName {  # 'mytunnel' should really
➥ be `basename $0` or some such
# for automagic config selection
type tun;
proto tcp;
keepalive yes;

pass $Secret;
```

```
up {
ifconfig "%% $LocalIP $PTP $RemoteIP arp";
route "add $TunnelHost $GW $GatewayIP";
route "delete default";
route "add default $GW $RemoteIP";
route "add $NET $LocalNetwork $GW $GatewayIP";
} ;

down {
ifconfig "%% down";
route "delete default";
route "delete $TunnelHost $GW $GatewayIP";
route "delete $NET $LocalNetwork";
route "add default $GW $GatewayIP";
} ;
}
```

# Rendszerfelügyelet

## 54–65. trükkök

Nehéz rájönni hogyan is kell egy rendszert jól beállítani, ha nem tudjuk, milyen módon viselkedik „normálisan". Okos kérdések felvetésével és a kapott válaszok helyes értelmezésével elkerülhetjük a változók körüli sötétben tapogatózást, és ott hajtunk végre hatékony módosításokat, ahol azok valóban helyénvalók.

Ez az a hely, ahol a naplóállományokat a legjobb barátainkként üdvözölhetjük. Bánjunk velük jól, szánjuk rájuk az őket megillető figyelmet: kötetnyi ismeretet meríthetünk belőlük arról hogyan is működik rendszerünk. De ha csupán hagyjuk, hogy naplóállományaink megtöltsék lemezeinket, akkor ez számos zavar forrása lehet. A megfelelő eszközökkel és módszerekkel a rendszernaplók átfogóak és elég részletesek lesznek ahhoz, hogy pontosan azt mutassák meg, amiről tudni szeretnénk.

Előfordul, hogy az adat, amelynek nyomába eredünk, semmilyen naplóba nem kerül rögzítésre, mindössze a futó rendszerben található meg, akár a rendszererőforrások hajszálpontos leltáraként vagy a hálózaton lévő adatként. Itt szeretnénk megjegyezni, hogy vizsgálatunk nem vállalkozik a teljes körű rendszerfelügyelet bemutatására,

vagy az olyan manapság divatos csomagok elemzésére, mint a Nagios vagy az MRTG. Ebben a fejezetben inkább azt mutatjuk be, hogyan faggathatjuk rendszerünket, hogy megtudjuk voltaképpen pontosan mi is zajlik benne.

Néhány módszert látni fogunk arra is, hogyan lehet a feltételezhető nehézségeket idejekorán érzékelni még a bekövetkeztük előtt, illetve miként lehet a végzetes rendszerleállásokat kezelni.

## #54 A syslog irányítása

### Hogyan lehet a syslogot keményebb munkára sarkallni, és miképpen lehet kevesebb időráfordítással áttekinteni hatalmas naplóállományokat?

A legtöbb Linux-változatban a *syslog* alapértelmezés szerinti telepítése nem igen végez jó munkát az adatosztályok külön-külön állományba gyűjtésével. Ha a */var/log/messages* állományban a Sendmailtől, `sudo`-tól, `bind`-tól és az egyéb rendszerszolgáltatásoktól származó üzenetek kusza halmazát látjuk, akkor valószínűleg felül kellene vizsgálni a */etc/syslog.conf* beállításfájlt.

Egy csomó szolgáltatás és fontossági szint létezik, amelyekre a *syslog* szűrést tud végezni:

| Szolgáltatások | Fontossági szint |
|---|---|
| auth | debug |
| auth-priv | info |
| cron | notice |
| daemon | warning |
| kern | err |
| lpr | crit |
| mail | alert |
| news | emerg |
| syslog | |
| user | |
| uucp | |
| local0–local7 | |

Vegyük figyelembe, hogy az alkalmazások maguk szokták eldönteni, milyen szolgáltatást, milyen fontossági szinttel kell a naplóban rögzíteni (a legjobbak lehetővé teszik a választást), éppen ezért előfordulhat az is, hogy nem az kerül a naplóba, amit várnánk. Az alábbiakban bemutatunk egy naplóállományt, amely „megpróbálja" összezavarni, mit hová is naplózott:

```
auth.warning /var/log/auth
mail.err /var/log/maillog
kern.* /var/log/kernel
cron.crit /var/log/cron
*.err;mail.none /var/log/syslog
*.info;auth.none;mail.none /var/log/messages

#*.=debug /var/log/debug

local0.info /var/log/cluster
local1.err /var/log/spamerica
```

Az előző sorok mindegyike a megadott, vagy annál nagyobb fontossági szint naplózását végzi a meghatározott állományokba. A *none* (semmi) kitüntetett fontosság arra utasítja a *syslog*-ot, hogy a megadott szolgáltatással egyáltalán ne foglalkozzon. A local0-tól local7-ig terjedő tartomány a saját programok rendelkezésére van bocsátva, tetszés szerint használható. Például a */var/log/spamerica* állományt a *local1.err* vagy annál magasabb fontosságú üzenetek töltik meg, amelyeket a levélszemétfeldolgozó program (spam processing job) hoz létre. Jó, ha ezek az üzenetek el vannak különítve az egyszerű levelezési naplótól (amelyek a */var/log/maillog*-ba kerülnek).

A megjegyzéssé tett `*.=debug` sor a démonként indított szolgáltatások kipróbálásánál bizonyul hasznosnak. Arra utasítja a *syslog*-ot, hogy csak a szolgáltatások *debug* (hibakereső) fontosságú üzeneteit naplózza; ezért általában nem is kell használni, hacsak nem bánjuk, hogy a lemezeinket hibakereső naplóállományok töltik meg. Egy másik megközelítés az, hogy a hibakereséssel kapcsolatos adatokat egy FIFO-tárolóba rögzítjük. Ilyen módon a naplóállományok nem fogják

felemészteni a helyet, hanem nyomtalanul eltűnnek, hacsak nem figyeli őket egy folyamat. FIFO-tárolóba történő naplózáshoz legelsőként létre kell hozni az állományrendszerben:

```
# mkfifo -m 0664 /var/log/debug
```

Aztán a *syslog.conf*-ban a `debug` bejegyzést ki kell egészíteni egy csővezeték jellel, valahogy így:

```
*.=debug |/var/log/debug
```

E beállítás nyomán a próbaüzem adatai a FIFO-tárolóban lesznek naplózva és a `less -f /var/log/debug` parancs segítségével lehet megtekinteni. Ez akkor is jól jön, ha az összes rendszerüzenetet egyetlen folyamattal figyeltetjük, amely elektronikus levélben figyelmeztet, amikor egy kényes üzenet érkezik.

Próbáljuk meg létrehozni a */var/log/monitor* nevű FIFO-tárolót, és a következő szabályhoz hasonlót adjuk a *syslog.conf*-hoz:

```
*.* |/var/log/monitor
```

Most valamennyi elsőbbségi sorba tartozó összes üzenet a */var/log/monitor* nevű FIFO-tárolóba kerül, és minden azt figyelő folyamat képes megfelelően válaszolni: mindez a lemezterület felhabzsolása nélkül zajlik.

### Ki az a Mark?

Figyeljük meg a */var/log/messages* állományban a következő sorokat!

```
Dec 29 18:33:35 catlin -- MARK --
Dec 29 18:53:35 catlin -- MARK --
Dec 29 19:13:35 catlin -- MARK --
Dec 29 19:33:35 catlin -- MARK --
Dec 29 19:53:35 catlin -- MARK --
Dec 29 20:13:35 catlin -- MARK --
Dec 29 20:33:35 catlin -- MARK --
Dec 29 20:53:35 catlin -- MARK --
Dec 29 21:13:35 catlin -- MARK --
```

Ezeket a sorokat a *syslog* jelző szolgáltatása hozza létre, afféle rendszer-működésjelző (touching base) gyanánt, úgyhogy elvileg ez alapján meg lehet mondani, hogy a *syslog* működése váratlanul ért-e véget. A legtöbb esetben ezek a jelzések csak a naplóállományok hizlalására jók, és amíg nincs gond a *syslog*-gal, valószínűleg nincs is szükség rájuk. Kikapcsolásához a `syslogd`-t lőjük ki, majd a `-m 0` kapcsolóval indítsuk újra:

```
# killall syslogd; /usr/sbin/syslogd -m 0
```

## Távoli naplózás

A `syslogd` korszerű változatai jó okból nem teszik lehetővé távoli gépek naplóbejegyzéseinek fogadását. Megfelelő óvintézkedések nélkül bármilyen rossz szándékú kívülálló telíthetné a gépet hamis *syslog* üzenetekkel, mivel nincs gép (host) hitelesítési lehetőség a *syslog*-ban. Mégis nagyon hasznos, ha központi *syslog*-unk van, különösen, ha egy csomó számítógépet kezelünk.

A fő gépen a távoli üzenetek fogadásához a `-r` kapcsolóval indítsuk a `syslogd`-t:

```
# /usr/sbin/syslogd -m 0 -r
```

Minden egyes gépen, ahonnan szeretnénk naplóbejegyzéseket kapni, a *syslog.conf*-ot ki kell egészíteni a következő sorral:

```
*.* @master.syslog.host.com
```

Természetesen az @ (kukac) karaktert követően a saját gépnevet vagy IP-címet kell megadni. Nagyon jó ötlet a syslog-kiszolgálón a syslog-kapu védelme. Ez az 514-es UDP-kapu, ami jól szűrhető a következő `iptables` paranccsal:

```
# iptables -A INPUT -t filter -p udp
➥ --dport 514 -s ! $LOCAL_NET -j DROP
```

Ebben a `$LOCAL_NET` jelöli azt a hálózatot vagy gépet, ahonnan a syslog-üzeneteket kapni szeretnénk (például 192.168.1.0/24 vagy *florian.nocat.net*).

Ha a rendszernaplók rendezettek, könnyebb a keresett adatok megtalálása. Ennek ellenére a rendszernaplók továbbra is terjengőssé válhatnak. A naplókkal kapcsolatban további útmutatást a [#75] „Színes naplóelemzés terminálablakban" trükkben olvashatunk. (Fontos megjegyezni, hogy a felsorolt képességek közül sokat alapértelmezésként lehetővé tesz több naplózó démon, például a `syslog-ng` – Lektor.)

## #55 Munkák figyelése a watch programmal

### Használjuk a watchot parancsok többszöri végrehajtására és az eredmények megtekintésére

Ha valaha akadt dolgunk a háttérben hosszan futó folyamattal, akkor valószínűleg a `ps` és `grep` parancsok többszöri használatával ellenőriztük, hogy vajon befejeződött-e már a futásuk:

```
mc@escher:~$ ps ax |grep tar
10303 ? S 0:00 bash -c cd /; tar cf - home
10304 ? S 0:42 tar cf - home
mc@escher:~$ ps ax |grep tar
10303 ? S 0:00 bash -c cd /; tar cf - home
10304 ? S 0:43 tar cf - home
```

Előfordult már, hogy `ftp`-vel töltöttünk le egy állományt, és mondjuk elfelejtettük kiadni a `hash` parancsot (és valamilyen okból kifolyólag nem használtuk az `ncftp`-t, `scp`-t, a `wget`-et vagy a `curl`-t), azaz fogalmunk sem volt róla, vajon mennyi időt fog elrabolni ez az átvitel. Ezt elkerülendő bejelentkeztünk újra és az állományt az `ls` paranccsal néztük meg:

```
mc@escher:~$ ls -l xfree86.tgz
-rw-r--r-- 1 rob users 12741812
➥ Jun 13 2001 xfree86.tgz
mc@escher:~$ ls -l xfree86.tgz
-rw-r--r-- 1 rob users 12744523
➥ Jun 13 2001 xfree86.tgz
```

Valahányszor azon kapjuk magunkat, hogy egy parancsot egymás után több alkalommal is kénytelenek vagyunk használni, próbáljuk ki a `watch` parancsot. Ez minden megadott parancsot folyamatosan is-

métel, tetszés szerinti gyakorisággal (az előre beállított érték 2 másodperc). Minden egyes alkalommal törli a képernyőt, így szép és könnyen áttekinthető felület áll rendelkezésre az olvasáshoz.

```
mc@escher:~$ watch 'ps ax |grep tar'

Every 2s: ps ax |grep tar|grep
➥ -v pts/0 Fri Sep 6 00:22:01 2002

10303 ? S 0:00 bash -c cd /; tar cf - home
10304 ? S 0:42 tar cf - home
```

Ha csővezetéket használunk, vagy az utasítás más védendő karaktereket tartalmaz – amelyeket nem szeretnénk, ha a `watch` futása előtt átalakulnának –, tegyük a parancsokat egyszeres idézőjelek (aposztrófok) közé. Az időtartam kijelölésére pedig a `-n` kapcsolót használjuk:

```
mc@escher:~$ watch -n1 ls -l xfree86.tgz

Every 1s: ls -l xfree86.tgz
➥ Fri Sep 6 00:31:41 2002

-rw-r--r-- 1 rob users 12756042
➥ Jun 13 2001 xfree86.tgz
```

A parancs minden menetben ki fogja emelni a változásokat, és hogy valóban kiugrók legyenek, a program invertált betűkkel jeleníti meg a változásokat. Próbáljuk ki a `-d` kapcsolót, ugyanis nagyon jól követhető a `'netstat -a | grep ESTAB'` parancs kimenetén, amint a hálózati kapcsolatok felépülnek és megszűnnek. Gondoljunk a `watch`-ra, mint kötelezően megszállott segítőtársunkra. A programból való kilépéshez nyomjuk le a CTRL+C billentyűket.

## Kapcsolódó anyag

- `man watch`

# #56 Mi tartja nyitva a kaput?

## A Netstat segítségével társítsuk könnyedén a kapuhoz az őt nyitva tartó folyamatot

A Netstat programmal könnyű listát készíteni a linuxos kiszolgálón figyelési állapotban levő kapukról:

```
root@catlin:~# netstat -ln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address
➥ Foreign Address State
tcp 0 0 0.0.0.0:5280 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN
tcp 0 0 10.42.3.2:53 0.0.0.0:* LISTEN
tcp 0 0 10.42.4.6:53 0.0.0.0:* LISTEN
tcp 0 0 127.0.0.1:53 0.0.0.0:* LISTEN
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN
udp 0 0 10.42.3.2:53 0.0.0.0:*
udp 0 0 10.42.4.6:53 0.0.0.0:*
udp 0 0 127.0.0.1:53 0.0.0.0:*
udp 0 0 0.0.0.0:67 0.0.0.0:*
raw 0 0 0.0.0.0:1 0.0.0.0:* 7
```

A szokásos szolgáltatások kapuit látjuk: a 80-ason a webkiszolgáló, az 53-ason a DNS, a 22-esen az `ssh` és a 67-esen a DHCP csücsül. De miféle folyamat várakozik figyelőállásban az 5280-as kapun?

A Netstat program mostani változatával egyszerű kitalálni, melyik program melyik kapuhoz kapcsolódik. Ha rendszergazdai azonosítóval vagyunk bejelentkezve a rendszerbe, akkor a program neve mellett használjuk a `-p`, vagyis a programokat jelölő kapcsolót:

```
root@catlin:~# netstat -lnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address
➥ Foreign Address State PID/Program name
tcp 0 0 0.0.0.0:5280 0.0.0.0:* LISTEN 698/perl
tcp 0 0 0.0.0.0:80 0.0.0.0:* LISTEN 217/httpd
tcp 0 0 10.42.3.2:53 0.0.0.0:* LISTEN 220/named
tcp 0 0 10.42.4.6:53 0.0.0.0:* LISTEN 220/named
```

```
tcp  0  0 127.0.0.1:53  0.0.0.0:*  LISTEN  220/named
tcp  0  0 0.0.0.0:22  0.0.0.0:*  LISTEN  200/sshd
udp  0  0 0.0.0.0:32768  0.0.0.0:*  220/named
udp  0  0 10.42.3.2:53  0.0.0.0:*  220/named
udp  0  0 10.42.4.6:53  0.0.0.0:*  220/named
udp  0  0 127.0.0.1:53  0.0.0.0:*  220/named
udp  0  0 0.0.0.0:67  0.0.0.0:*  222/dhcpd
raw  0  0 0.0.0.0:1  0.0.0.0:*  7  222/dhcpd
```

Igen, így már jobb. A PID 698 egy Perl-folyamat, amely az 5280-as kapuhoz kapcsolódik. Most megfogjuk a `ps`-sel:

```
root@catlin:~# ps auwex |grep -w 698
nocat  698  0.0  2.0  5164  3840 ?  S
➥ Aug25  0:00 /usr/bin/perl -w ./bin/gateway
➥ PWD=/usr/local/nocat HOSTNAME=catlin.r
```

A `ps auwex` parancs a következő kapcsolókkal használható:

- `-a` megmutat nekünk minden folyamatot;
- `-x` a nem párbeszédeseket;
- `-u` a felhasználói adatokkal kiegészített folyamatokat;
- `-w` részletes lista formában;
- `-e` környezeti bitekkel kiegészítve.

Aztán `grep`-pel rákeresünk a szóhatárokon a PID-re. Most már tisztább a kép és tudjuk, hogy a */usr/local/nocat* könyvtárhoz tartozó, *nocat* azonosítójú felhasználó futtatja a */bin/gateway*-t, egy olyan Perl-folyamatot, amely az 5280-as kapun csücsül. A Netstattal a `-p` kapcsoló nélkül egy nyitott kapuhoz kapcsolódó adott folyamat megtalálása sokkal bonyolultabb.

Ha véletlenül egyszerű felhasználói jogosultságokkal rendelkezünk (elfelejtettünk rendszergazdaként bejelentkezni), akkor a rendszer nem fogja megmutatni, melyik program milyen kapun fut. Ehelyett az alábbihoz hasonló hibaüzent fog megjelenni:

```
(No info could be read for "-p":
➥ geteuid(  )=1000 but you should be root.)
```

Ekkor jelentkezünk be újra, de rendszergazdaként, vagy olvassuk el a [#12] „Dolgozzon a sudo!" trükköt, hogy megtudjuk milyen módon kell a sudo-val a rendszergazdai szerepkört meglepően jól utánozni.

## #57 Nyitott állományok és foglalatok ellenőrzése az lsof-fal

### Könnyen megtudhatjuk, milyen állományokat, könyvtárakat és foglalatokat tartanak nyitva a folyamatok

Próbálkoztunk-e valaha leválasztani egy állományrendszert csupán azért, hogy megtudjuk, vajon használja-e még valamelyik folyamat?

```
root@mouse:~# umount /mnt
umount: /mnt: device is busy
```

Ahhoz, hogy gyorsan kiderítsük mely programok használják még mindig a */mnt* könyvtárat, alkalmazzuk az *lsof* eszközt:

```
root@mouse:~# lsof /mnt
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
bash 30951 rob cwd DIR 7,0 1024 2 /mnt
```

Nahát, rob egy találomra kiadott cd paranccsal éppen a */mnt*-be lépett be (mivel az ő *bash* folyamatában ez lett beállítva *cwd* könyvtárnak). Az *lsof* fel fogja sorolni a meghatározott folyamathoz tartozó nyitott állományokat, könyvtárakat, folyamatokat és készülékeket. A fenti példában megadtunk egy befűzési pontot az *lsof*-nek, az pedig megmutatta a kapcsolódó folyamatokat. Fordított lekérdezéshez, vagyis amikor az érintett állományokhoz tartozó PID-et keressük, a -p kapcsolót használjuk:

```
root@mouse:~# lsof -p 30563
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
inetd 30563 root cwd DIR 3,3 408 2 /
inetd 30563 root rtd DIR 3,3 408 2 /
inetd 30563 root txt REG 3,3 21432 39140
➥ /usr/sbin/inetd
inetd 30563 root mem REG 3,3 432647 11715
➥ /lib/ld-2.2.3.so
```

```
inetd 30563 root mem REG 3,3 4783716 11720
➥ /lib/libc-2.2.3.so
inetd 30563 root mem REG 3,3 19148 11708
➥ /lib/libnss_db-2.2.so
inetd 30563 root mem REG 3,3 238649 11728
➥ /lib/libnss_files-2.2.3.so
inetd 30563 root mem REG 3,3 483324 11710
➥ /lib/libdb-3.1.so
inetd 30563 root 0u CHR 1,3 647 /dev/null
inetd 30563 root 1u CHR 1,3 647 /dev/null
inetd 30563 root 2u CHR 1,3 647 /dev/null
inetd 30563 root 4u IPv4 847222
➥ TCP *:telnet (LISTEN)
inetd 30563 root 5u IPv4 560439
➥ TCP *:cvspserver (LISTEN)
```

Ha a folyamatot név szerint szeretnénk megadni, használjuk a `-c` kapcsolót:

```
root@mouse:~# lsof -c syslogd
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
syslogd 25627 root cwd DIR 3,3 408 2 /
syslogd 25627 root rtd DIR 3,3 408 2 /
syslogd 25627 root txt REG 3,3 27060 5538
➥ /usr/sbin/syslogd
syslogd 25627 root mem REG 3,3 432647 11715
➥ /lib/ld-2.2.3.so
syslogd 25627 root mem REG 3,3 4783716 11720
➥ /lib/libc-2.2.3.so
syslogd 25627 root mem REG 3,3 238649 11728
➥ /lib/libnss_files-2.2.3.so
syslogd 25627 root mem REG 3,3 75894 11719
➥ /lib/libnss_dns-2.2.3.so
syslogd 25627 root mem REG 3,3 225681 11724
➥ /lib/libresolv-2.2.3.so
syslogd 25627 root mem REG 3,3 19148 11708
➥ /lib/libnss_db-2.2.so
syslogd 25627 root mem REG 3,3 483324 11710
➥ /lib/libdb-3.1.so
syslogd 25627 root 0u unix 0xdcc2d5b0 775254
➥ /dev/log
```

```
syslogd 25627 root 1w REG 3,3 135744 5652
➥ /var/log/debug
syslogd 25627 root 2w REG 3,3 107459 5651
➥ /var/log/syslog
syslogd 25627 root 3w REG 3,3 107054 58317
➥ /var/log/maillog
syslogd 25627 root 4w REG 3,3 4735 16
➥ /var/log/authlog
```

A parancssorban különleges eszközt is meghatározhatunk. Lássuk például, mire is készül a felhasználó a `pts/0`-on:

```
rob@mouse:~# lsof /dev/pts/0
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
bash 29816 rob 0u CHR 136,0 2 /dev/pts/0
bash 29816 rob 1u CHR 136,0 2 /dev/pts/0
bash 29816 rob 2u CHR 136,0 2 /dev/pts/0
bash 29816 rob 255u CHR 136,0 2 /dev/pts/0
lsof 30882 root 0u CHR 136,0 2 /dev/pts/0
lsof 30882 root 1u CHR 136,0 2 /dev/pts/0
lsof 30882 root 2u CHR 136,0 2 /dev/pts/0
```

Ha több kapcsolót kell megadnunk, ezek alapértelmezésben logikai VAGY kapcsolatban állnak egymással. Ha nem ez a szándékunk, akor mindegyik elé a `-a` zászlót (flag) kell illeszteni, ami a logikai ÉS kapcsolatot fogja eredményezni. Például, ha szereténk látni az összes nyitott fájlt, amit rob `vi` folyamatához kapcsolódik, próbáljuk meg ezt:

```
root@mouse:~# lsof -u rob -ac vi
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
vi 31059 rob cwd DIR 3,3 2824 39681 /home/rob
vi 31059 rob rtd DIR 3,3 408 2 /
vi 31059 rob txt REG 3,3 554504 9799 /usr/bin/vim
vi 31059 rob mem REG 3,3 432647 11715
➥ /lib/ld-2.2.3.so
vi 31059 rob mem REG 3,3 282178 2825
➥ /lib/libncurses.so.5.2
vi 31059 rob mem REG 3,3 76023 2831
➥ /usr/lib/libgpm.so.1.18.0
vi 31059 rob mem REG 3,3 4783716 11720
➥ /lib/libc-2.2.3.so
```

```
vi 31059 rob mem REG 3,3 249120 11721
➥ /lib/libnss_compat-2.2.3.so
vi 31059 rob mem REG 3,3 357644 11725
➥ /lib/libnsl-2.2.3.so
vi 31059 rob 0u CHR 136,1 3 /dev/pts/1
vi 31059 rob 1u CHR 136,1 3 /dev/pts/1
vi 31059 rob 2u CHR 136,1 3 /dev/pts/1
vi 31059 rob 3u REG 3,3 4096 15
➥ /home/rob/.sushi.c.swp
```

Ha szeretnénk megtudni mely foglalatok nyitottak, és milyen folyamatok kapcsolódnak hozzájuk (ahogyan ezt a Netstat `-p` használatánál láttuk), akkor próbáljuk ki a `-i` kapcsoló használatát:

```
rob@mouse:~# lsof -i
COMMAND PID USER FD TYPE DEVICE SIZE NODE NAME
sshd 69 root 3u IPv4 61 TCP *:ssh (LISTEN)
mysqld 126 mysql 3u IPv4 144 TCP *:3306 (LISTEN)
mysqld 128 mysql 3u IPv4 144 TCP *:3306 (LISTEN)
mysqld 129 mysql 3u IPv4 144 TCP *:3306 (LISTEN)
httpd 24905 root 22u IPv4 852520 TCP *:443 (LISTEN)
httpd 24905 root 23u IPv4 852521 TCP *:www (LISTEN)
httpd 28383 www 4u IPv4 917713
➥ TCP nocat.net:www->65.192.187.158:8648
➥ (ESTABLISHED)
httpd 28389 www 4u IPv4 917714
➥ TCP nocat.net:www->65.192.187.158:9832
➥ (ESTABLISHED)
httpd 28389 www 22u IPv4 852520 TCP *:443 (LISTEN)
httpd 28389 www 23u IPv4 852521 TCP *:www (LISTEN)
exim 29879 exim 0u IPv4 557513 TCP *:smtp (LISTEN)
inetd 30563 root 4u IPv4 847222
➥ TCP *:telnet (LISTEN)
inetd 30563 root 5u IPv4 560439
➥ TCP *:cvspserver (LISTEN)
sshd 30973 root 4u IPv4 901571
➥ TCP nocat.net:ssh->some.where.net:52543
➥ (ESTABLISHED)
sshd 30991 rob 4u IPv4 901577
➥ TCP nocat.net:ssh->some.where.else.net:52544
➥ (ESTABLISHED)
```

Sohase feledkezzünk meg arról, hogy az *lsof* számos szolgáltatásának igénybevételéhez rendszergazdaként (root) kell bejelentkezni, ide sorolandó a nyitott foglalatok lekérdezése is. Az *lsof* összetett, de egyben rendkívül rugalmas eszköz, amely éppen annyi adatot tud szolgáltatni a rendszeren futó folyamatok által használt fájlokról, amennyire szükségünk van.

### Kapcsolódó anyag

- Az lsof legfrissebb változata az `ftp://vic.cc.purdue.edu/pub/tools/unix/lsof/` címről tölthető le.

## #58 Rendszererőforrások felügyelete top-pal

### A rendszer tevékenységének jobb áttekintésére használjunk top segédprogramot

A `top` parancs a második legjobb rendszerterhelés-, memóriahasználat- és processzorterhelés-jelző. A *procps* csomag tartalmazza. A *top* segédprogramot legegyszerűbben a parancssorból indíthatjuk:

```
$ top
```

Két másodperces frissítési időközönként teljes képernyőnyi adatot kapunk:

```
3:54pm up 1 day, 16 min, 2 users,
➥ load average: 0.00, 0.00, 0.00
38 processes: 37 sleeping, 1 running,
➥ 0 zombie, 0 stopped
CPU states: 0.0% user, 0.7% system,
➥ 0.0% nice, 99.2% idle
Mem: 189984K av, 155868K used, 34116K free,
➥ 0K shrd, 42444K buff
Swap: 257032K av, 0K used,
➥ 257032K free 60028K cached
```

```
PID USER PRI NI SIZE RSS SHARE STAT
➥ %CPU %MEM TIME COMMAND
6195 rob 14 0 1004 1004 800 R 0.5 0.5 0:00 top
1 root 8 0 212 212 180 S 0.0 0.1 0:13 init
2 root 9 0 0 0 0 SW 0.0 0.0 0:00 keventd
3 root 9 0 0 0 0 SW 0.0 0.0 0:00 kswapd
4 root 9 0 0 0 0 SW 0.0 0.0 0:00 kreclaimd
5 root 9 0 0 0 0 SW 0.0 0.0 0:00 bdflush
6 root 9 0 0 0 0 SW 0.0 0.0 0:00 kupdated
8 root -1 -20 0 0 0 SW< 0.0 0.0 0:00 mdrecoveryd
176 root 9 0 788 788 680 S 0.0 0.4 0:00 syslogd
179 root 9 0 1228 1228 444 S 0.0 0.6 0:00 klogd
182 root 8 0 1228 1228 1104 S 0.0 0.6 0:06 sshd
184 root 8 0 616 616 520 S 0.0 0.3 0:00 crond
186 daemon 9 0 652 652 560 S 0.0 0.3 0:00 atd
197 root 9 0 2544 2544 2396 S 0.0 1.3 0:00 httpd
200 root 9 0 3740 3740 1956 S 0.0 1.9 0:00 named
202 root 9 0 1004 1004 828 S 0.0 0.5 0:00 dhcpd
203 root 9 0 504 504 444 S 0.0 0.2 0:00 agetty
```

Ha lenyomjuk a ? (kérdőjel) billentyűt, a használható parancsok listájához jutunk. Hadd említsünk meg e helyen néhány hasznos billentyűt:

M – állandó jelleggel tárban levő programok mérete;
P – a processzorhasználat alapján rendezett lista;
S – halmozott futásidő (cumulative runtime), vagyis mennyi ideje tart az egyes folyamatok vagy gyermek-folyamataiknak a végrehajtása processzor-másodpercekben kifejezve;
i – az éppen üresjáratban levő folyamatok megjelenítésének leállítása.

Amennyiben rendszergazdaként adtuk ki a `top` parancsot, adódik néhány további párbeszédes üzemmódú és rendezőparancs, ezek alkalmazása munkánk során hasznos lehet. Az U billentyű lehetővé teszi valamennyi folyamat kiszűrését, kivéve az adott felhasználóhoz tartozókat. Ezt követően használható a K, amely párbeszédes formában lehetővé teszi egy megadott PID kilövését (`kill`) – tetszőleges leállító jel használatával. Ez különösen az elszabadult folyamatok leállításakor bizonyulhat hasznosnak, amelyeket a rettegett elgépelések elkerülése végett közvetlenül a `top`-ból lehet kiirtani, a PID másolása, illetve beillesztése révén.

A `top` parancsot célszerű állandóan futtatni a nagy kihasználtságú gépeken, amelyekre ugyan bejelentkeztünk, de amelyek egyébként nem dolgoznak. Ha szeretnénk folyamatosan látni a leterheltség mértékét a bejelentkező ablak címsorában – afféle *mini.top*-ot, miközben a saját héjprogramunkban maradunk –, akkor tanulmányozzuk az [#59] „Az átlagos terhelés folyamatos megjelenítése a címsorban" című trükköt.

### Kapcsolódó anyagok

- [#17] Folyamatok közvetett felügyelete a *procps* segítségével
- [#59] Az átlagos terhelés folyamatos megjelenítése a címsorban

## #59 Az átlagos terhelés folyamatos megjelenítése a címsorban

### Avagy hogyan könnyítsük meg feladatunkat a címsor munkába fogásával?

Amennyiben már régebb óta felügyelünk egy linuxos kiszolgálót, bizonyára közelebbi kapcsolatba kerültünk a *top* segédprogrammal. Ha viszont még nem, sietve gépeljük be a `top` parancsot a legközelebbi konzolablakba, és amikor kezdünk ráunni az adatáradatra, nyomjunk egy ? (kérdőjel) billentyűt.

Ha viszont már több linuxos kiszolgáló felügyeletét is elláttuk, valószínűleg van fogalmunk arról, milyen is az, amikor több ablakban fut a `top`, s mindegyik egyformán verseng a rendszer erőforrásaiért.

A számítástechnika világában az elpazarolt erőforrásokat jobban is fel lehetett volna használni, hogy több segítséget nyújtsanak a felhasználónak. Miért ne futtathatnánk egy olyan folyamatot, amely valós időben a konzol címsorába ki tudja írni a rendszerterhelést, tekintet nélkül arra, milyen más alkalmazások működnek közben?

Mentsük *tl* néven az alábbi héjprogramot a *~/bin/* könyvtárba:

### Lista: tl

```
#!/usr/bin/perl -w

use strict;
$|++;

my $host=`/bin/hostname`;
chomp $host;

while(1) {

open(LOAD,"/proc/loadavg") ||
➥ die "Couldn't open /proc/loadavg: $!\ n";

my @load=split(/ /,<LOAD>);
close(LOAD);

print "\ 033]0;";
print "$host: $load[0] $load[1] $load[2] at ",
➥ scalar(localtime);
print "\ 007";

sleep 2;
}
```

Ha azt szeretnénk, hogy a gépünk címsorában megjelenjen a gép neve, az átlagos terhelés mértéke és a pillanatnyi idő, akkor gépeljük be a `tl&` parancsot. Remekül fut a háttérben, még akkor is, ha olyan felhasználói beavatkozást igénylő (interactive) programot használunk, mint a `vim`. De az sem okoz gondot, ha a kiírandó adatok közé még a pillanatnyi munkakönyvtárat is felveszi. Amikor könyvtárat váltunk a `cd`-vel, ez egy pillanatra felvillan a címsorban, aztán helyette ismét az idő és az átlagos terhelés jelenik meg.

Ahelyett azonban, hogy mindenhová terminálokat helyeznénk, és konzolokkal borítanánk be munkaasztalunkat, ezeket felhalmozhatjuk oly módon, hogy csak a címsoruk maradjon látható, így egyetlen pillantással áttekinthető, hogyan is dolgoznak a gépek. Amint valamelyik gépen tennivaló akad, akkor azt az ablakot hozzuk előtérbe, amelyik szükséges, és már kezdhetünk is dolgozni. Mindez elérhető úgy, hogy nem kell telepíteni semmiféle programot a helyi gépre.

Amint befejeztük a munkát, a kilépés előtt ne felejtsük el a `killall tl` paranccsal kilőni a programot.

A lustaság fél egészség, tehát próbáljuk ki a következő parancsot is:

```
$ $ echo 'killall tl > /dev/null 2>&1'
➥ >> ~/.bash_logout
```

Ez a műveletsorozat kilépéskor mindegyik `tl` munkát kilövi, egyetlen fölösleges további mozdulat nélkül. Egyre könnyebb lesz a rendszerfelügyelet, nemigaz?

## #60 Hálózatfelügyelet ngrep-pel

### Lássuk csak a ki, mit csinál műveleteket a greppel a hálózati csatolón

Az `ngrep` egy érdekes csomagbefogó eszköz, amely hasonlít a `tcpdump`-hoz, illetve a `snoop`-hoz. Ez a program abból a szempontból egyedi, hogy megpróbálja a lehetőségek szerint legkönnyebbé tenni a befogott csomagok különböző szempontoknak való megfelelését, például mely csomagok tartalmát kell kinyomtatni. A program `grep`-megfelelő formátumot használ, amit szabályos kifejezések és egy sor GNU-grep kapcsoló tesz teljessé. Nyomtatás előtt a csomagokat ASCII-kódúvá vagy hexadecimálissá – tizenhatos számrendszerbeli alakúvá – alakítja.

A példa kedvéért az útválasztónkon a HTTP `GET`-kérések nyomán áthaladó teljes HTTP-forgalom tartalmának megtekintéséhez a következőket próbáljuk meg alkalmazni:

```
# ngrep -q GET
```

Ha egy bizonyos gépre, protokollra, kapura vagy más csomagszűrő szempontra vagyunk kíváncsiak, ezt kijelölhetjük egy *bpf* szűrővel, vagy akár adatsémával.

A parancs felépítése a tcpdump-éhoz hasonló:

```
# ngrep -qi rob@nocat.net port 25

T 10.42.4.7:65174 -> 209.204.146.26:25 [AP]
RCPT TO:<rob@nocat.net>..

T 209.204.146.26:25 -> 10.42.4.7:65174 [AP]
250 2.1.5 <rob@nocat.net>... Recipient ok..

T 10.42.4.7:65174 -> 209.204.146.26:25 [AP]
Date: Sun, 8 Sep 2002 23:55:18
➥ -0700..Mime-Version: 1.0 (Apple Message fram
ework v543)..Content-Type: text/plain;
➥ charset=US-ASCII; format=flowed..Sub
ject: Greetings.....From: John Doe
➥ <johnd@somewhere.else.com>..To: rob@noca
t.net..Content-Transfer-Encoding:
➥ 7bit..Message-Id: <19DB8C16-C3C1-11D6-B23
9-0003936D6AE0@somewhere.else.com>..X-Mailer:
➥ Apple Mail v2)....What does t
hat pgp command you mentioned do
➥ again?....Thanks,....--A Friend....
```

Minthogy az ngrep az STDOUT szabályos kimenetre nyomtat, így a kimeneten a felhasználó utófeldolgozást is végezhet. Ha a kimenetet csak a magunk számára dolgozzuk fel, akkor használjuk a -l kapcsolót, amellyel gyorstározottá tettük. Például, ha azt szeretnénk megtudni, hogy kik és mit keresnek a hálózatra csatlakozva, megtudhatjuk, ha kipróbáljuk ezt a kis Perl nyelven íródott programocskát:

### Lista: go ogle

```
#!/usr/bin/perl
use Socket;
$|++;

open(NG,"ngrep
➥ -lqi '(GET|POST).*/(search|find)' |");
print "Go ogle online.\ n";
my ($go,$i) = 0;
my %host = ( );
```

```
while(<NG>) {

if(/^T (\ d+\ .\ d+.\ d+\ .\ d+):\
➥ d+ -> (\ d+\ .\ d+\ .\ d+\ .\ d+):80/) {
$i = inet_aton($1);
$host{ $1}  ||= gethostbyaddr($i, AF_INET) || $1;
$i = inet_aton($2);
$host{ $2}  ||= gethostbyaddr($i, AF_INET) || $2;
print "$host{ $1}  -> $host{ $2}  : ";
$go = 1;
next;
}
if(/(q|p|query|for)=(.*)?(&|HTTP)/) {
next unless $go;
my $q = $2;
$q =~ s/(\ +|&.*)/ /g;
$q =~ s/%(\ w+)/chr(hex($1))/ge;
print "$q\ n";
$go = 0;
}
}
```

A héjprogramnak a go-ogle nevet adtam: egy `ngrep` futtatásával megkeres bármilyen `GET`- vagy `POST`-kérést, amely valahol a címben `search` (keres) vagy `find` (talál) utasításokat tartalmaz. Az eredmény az alábbi néhány sorhoz fog hasonlítani:

```
Google online.
caligula.nocat.net -> www.google.com
➥ : o'reilly mac os x conference
caligula.nocat.net -> s1.search.vip.scd.yahoo.com
➥ : junk mail $$$
tiberius.nocat.net -> altavista.com : babel fish
caligula.nocat.net -> 166-140.amazon.com : Brazil
livia.nocat.net -> 66.161.12.119 : lart
```

A program nem fogja kikerülni a kérésben szereplő kódolt karakterláncokat sem: figyeljük csak meg a `google` lekérdezésben az ' egyszeres idézőjelet (aposztrófot ), a yahoo-nál pedig a dollár karaktereket ($$$). Az IP-címeket gépnevekké alakítja – minthogy az `ngrep`, úgy tűnik, nem képes erre, ezért a legkedvezőbb jellemzőket a se-

besség terén fogja felmutatni. Érdekességszámba megy az utolsó két eredmény: a brazil lekérdezés valójában a `http://www.imdb.com` címen futott, az utolsó pedig a `http://www.dictionary.com` címen. Nyilvánvaló, hogy az IMDB partneri kapcsolatban áll az Amazonnal, és hogy a Dictionary.com keresőgépének nincs PTR-bejegyzése. Meglepő milyen sokat lehet tanulni más emberek csomagjait vizsgálgatva.

Fontos megjegyeznünk: az `ngrep` futtatásához rendszergazdaként kell belépni, és a legjobb eredmények eléréséhez a hálózat szélén levő útválasztóról kell indítani.

### Kapcsolódó anyagok

- `man ngrep`
- `http://www.packetfactory.net/Projects/ngrep/`

## #61 Saját gépek átvizsgálása az nmap-pel

### Járjunk utána, mely kiszolgálók és szolgáltatások érhetők el a hálózaton!

Ha ezelőtt még sohasem használtuk volna, elárulom az `nmap` kifejezetten hasznos segédeszköz a hálózatra kapcsolt gépek és szolgáltatások azonosításában. Sokfajta hálózati letapogatást képes végrehajtani: a szabályos TCP/UDP-től a különlegesebb rejtett TCP SYN-letapogatásokon át a karácsonyfa és `Null` próbáig és még sorolhatnám.

Talán még ennél is érdekesebb az operációsrendszer-lenyomatkód, amely a célgép által visszaküldött csomagokat elemzi és hasonlítja össze az eredményt az ismert operációs rendszerek adatbázisával. Ez egy lenyűgözően izgalmas szolgáltatás, minthogy képes azonosítani a túloldalon levő operációs rendszer típusát anélkül, hogy annak valamilyen szolgáltatását igénybe kellene vennie, sőt a vizsgált gépről még a becsült folyamatos üzemidőt is megjeleníti.

Egy operációsrendszer-lenyomatellenőrzéssel bővített általános kapupásztázáshoz használjuk a `-O` kapcsolót:

```
rob@catlin:~# nmap -O caligula

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on
➥ caligula.rob.nocat (10.42.4.7):
(The 1600 ports scanned but not shown
➥ below are in state: closed)
Port State Service
22/tcp open ssh
Remote operating system guess:
➥ Mac OS X 10.1 - 10.1.4
Uptime 5.760 days (since Tue Sep 3 19:14:36 2002)

Nmap run completed -- 1 IP address
➥ (1 host up) scanned in 31 seconds
```

Ha nmap-pel szeretnénk teljes hálózatunkat átvizsgálni és az időnk engedi, a parancssorban megadhatunk egy hálózatot és alhálózatot. A következő parancs TCP SYN-letapogatást és lenyomatellenőrzést végez az első 64 címen a 10.42.4.0 alhálózaton:

```
root@catlin:~# nmap -OsS 10.42.4.0/26
```

Mivel az nmap a STDOUT szabályos kimenetre küldi az eredményeket, így ezek a letapogatási eredmények állományba menthetők és a korábbi futtatások eredményével könnyedén összevethetők, továbbá akár jelentés is készíthető az eltérésről. Egy karácsonyfa letapogatást fogunk futtatni és a grep-pel kiemelünk néhány sort – mint például a futási idő (run time) – a hibás találatok kiszűrésére:

```
root@catlin:~# nmap -sX 10.42.4.0/26 |
➥ egrep -v '^(Nmap|Starting)' \
  > nmap.output
```

Most hajtsuk végre ugyanazt a parancsot még egyszer, mondjuk másnap, véletlenszerűen kiválasztott órában:

```
root@catlin:~# nmap -sX localhost |
➥ egrep -v '^(Nmap|Starting)' \
  > nmap.output2
```

Azután a `diff` segítségével nézzük meg, mi változott:

```
root@catlin:~# diff -c nmap.output*
*** nmap.output Mon Sep 9 14:45:06 2002
--- nmap.output2 Mon Sep 9 14:45:21 2002
***************
*** 1,7 ****

Interesting ports on catlin.rob.nocat (10.42.4.6):
! (The 1598 ports scanned but not shown below are
   ➥ in state: closed)
Port State Service
22/tcp open ssh
53/tcp open domain
80/tcp open http
--- 1,8 ----

Interesting ports on catlin.rob.nocat (10.42.4.6):
! (The 1597 ports scanned but not shown below are
   ➥ in state: closed)
Port State Service
+ 21/tcp open ftp
22/tcp open ssh
53/tcp open domain
80/tcp open http
root@catlin:~#
```

Ez lenyűgöző: úgy néz ki, hogy a catlin egy bizonyos időpontban felvette a kapcsolatot egy FTP-kiszolgálóval. Ez a módszer minden futás alkalmával képes megtalálni új és már nem élő kapcsolatokat gépekkel és szolgáltatásokkal. A `nmap` kimenetéből képzett dokumentumgyűjteményben – dátummal és időponttal megjelölt állományokban vagy adatbázisban – naplót lehet vezetni az összes hálózatba kötött gép állapotáról. A program héjprogramba foglalása és `cron`-ból történő indítása már csak gyakorló feladat, amely remélhetőleg örömet fog szerezni, és egészen biztosan megéri elvégezni.

## Kapcsolódó anyag

- Az nmap honlapja: `http://www.insecure.org/nmap`

# #62 Lemezavulás-elemzés (disk age analysis)

## Könnyen azonosítható, hogy a lemez mely részei változnak gyakran

Hogyan lehet hamarjában megmondani, hogy az állományrendszer mely részei frissülnek gyakran, és melyek azok, amelyek már hónapok óta semmit sem változtak? Játszi könnyedséggel megadható a válasz a megfelelő Perl-program alkalmazásával.

Az alábbiakban látható egy Perl-héjprogram, amely az állományrendszeren avuláselemzést végez. A program a lemezterületet két szempont szerint osztja fel, az utolsó módosítás dátuma és az utolsó hozzáférés dátuma szerint. A program próbafuttatása a következő eredményeket hozta:

```
% diskage /usr/local

Disk aging analysis for /usr/local:

last num last num
Age (days) modified files accessed files
0 - 30 260403 Kb 817 140303 Kb 6968
31 - 60 11789 Kb 226 23140 Kb 199
61 - 90 40168 Kb 1126 1087585 Kb 31625
91 - 180 118927 Kb 995 0 Kb 0
181 - 365 85005 Kb 1889 0 Kb 0
366 - 9999 734735 Kb 33739 0 Kb 0
----------- ----- ----------- -----
Total 1251029 Kb 38792 1251029 Kb 38792
```

A héjprogramot a -v kapcsolóval futtatva készíthetjük el az legutóbb módosított állományokat, és a legutolsó hozzáférés dátumát tartalmazó listát.

### Lista: diskage

```
#!/usr/local/bin/perl
#
# Disk aging report generator
# Written by Seann Herdejurgen
#
# May 1998
```

```
use File::Find;

# Initialize variables
@levels=(30,60,90,180,365,9999);

# Check for verbose flag
if ($ARGV[0] eq "-v") {
$verbose++;
shift(@ARGV);
}

$ARGV[0]=$ENV{ 'PWD'}  if ($ARGV[0] eq "");

foreach $dir (@ARGV) {
foreach $level (@levels) {
$modified{ $level} =0;
$accessed{ $level} =0;
$mfiles{ $level} =0;
$afiles{ $level} =0;
}
print("\ nDisk aging analysis for $dir:\ n\ n");
print (" mod acc size file\ n") if ($verbose);

# Traverse desired filesystems
find(\ &wanted,$dir);

print(" last num last num\ n");
print(" Age (days) modified files
➥ accessed files\ n");
$msize=$asize=$mtotal=$atotal=$lastlevel=0;

foreach $level (@levels) {
printf("%4d - %4d %8d Kb %5d %8d Kb %5d\ n",
➥ $lastlevel,$level,$modified{ $level} /1024,
➥ $mfiles{ $level} ,$accessed{ $level} /1024,
➥ $afiles{ $level} );
$msize+=$modified{ $level} /1024;
$asize+=$accessed{ $level} /1024;
$mtotal+=$mfiles{ $level} ;
$atotal+=$afiles{ $level} ;
$lastlevel=$level+1;
}
```

```
printf(" ----------- ----- ----------- -----\ n");
printf(" Total %8d Kb %5d %8d Kb %5d\ n",
➥ $msize,$mtotal,$asize,$atotal);
}

exit;

sub wanted {
(($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size) =
➥ lstat($_));
$mod=int(-M _);
$acc=int(-A _);
foreach $level (@levels) {
if ($mod<=$level) {  $modified{ $level} +=$size;
➥ $mfiles{ $level} ++; last; }
}
foreach $level (@levels) {
if ($acc<=$level) {  $accessed{ $level} +=$size;
➥ $afiles{ $level} ++; last; }
}
printf("%4d %4d %6d %s\ n",$mod,$acc,$size,$_)
➥ if ($verbose);
}
```

# #63 IP-átvétel kevés ráfordítással

## IP-átvétel pinggel, bashsel és egy egyszerű hálózati segédprogrammal

Több gép egyikéhez az adatforgalom átirányítása meglehetősen egyszerű a címfogatásos névkiszolgáló (round-robin DNS) használatával, amint azt a [#79] „Terhelésmegosztás címforgatásos DNS segítségével" részben ismertetjük. De vajon mi történik akkor, ha e kiszolgálók egyike elérhetetlenné válik? A következőkben bemutatunk egy elgondolást, hogyan lehet egy másik kiszolgáló állapotát felügyelni, illetve annak meghibásodása esetén miképpen lehet a helyére állítani egy másikat. (Az itt bemutatandó megoldásról érdemes tudni, hogy csupán kezdetleges megoldás egy igazi HA-rendszerhez képest, valamint hogy a tűzfalunkat is fel kell készítenünk erre az átállásra – Lektor.)

Először is különbséget kell tennünk egy kiszolgáló valódi IP-címe és az IP-cím (vagy címek) között, ahonnan a nyilvános tartalmat ténylegesen szolgáltatja. Így a példa kedvéért két kiszolgálóra fogunk hivatkozni:

pinkyre és brainre. Pinky az `eth0` hálózati csatolón a 208.201.239.12-es „valódi" IP-címet használja, de van egy másodcíme (alias) is az `eth0:0` felületen: a 208.201.239.36-os. Hasonlóképpen a brain nevű kiszolgáló a 208.201.239.13-as címet fogja „valódi" címként használni, és másodcímként az `eth0:0` felületén a 208.201.239.37-et.

Amennyiben még sohasem használtunk IP-másodcímeket, akkor most következzék egy gyorstalpaló HOGYAN:

```
# ifconfig eth0:0 1.2.3.4
```

Máris van egy új IP-címünk – az 1.2.3.4 –, amely az `eth0` csatolófelülethez kapcsolódik, de ezt `eth0:0`-nak hívják. Korábban külön be kellett kapcsolni az *IP-alias* szolgáltatást a rendszermag fordításakor, úgy tűnik ez a beállítási lehetőség eltűnt a legújabb rendszermagokból, de magát a szolgáltatást mindenképpen tartalmazza. Egy dolgot sosem szabad elfelejteni az IP-másodcím szolgáltatás kapcsán: ha azt a csatolófelületet, amelyhez IP-másodcímek kapcsolódnak leállítják, akkor egyidejűleg valamennyi IP-másodcím leáll. A másodcímnek számokból és betűkből álló (alfanumerikus) nevet is adhatunk, azonban lehet, hogy az `ifconfig` a csatolófelületek listázásakor a névnek csak az első négy-öt karakterét fogja megjeleníteni.

Ha már a pinkyn és a brainen egyaránt beállítottuk az `eth0:0`-t, akkor kössünk egy szolgáltatást a másod-IP-hez, mondjuk az Apache webkiszolgálót, és állítsuk be a címforgatásos névkiszolgálót olyan módon, hogy egyetlen gépnévvel hivatkozzon mindkettőre (lásd [#79] Terhelésmegosztás címforgatásos DNS segítségével). Feltételezni fogjuk, hogy a *www.oreillynet.com* számára tartalék webszolgáltatást fogunk kiépíteni, amelynek névfeloldása a 208.201.239.36 vagy 208.201.239.37-es IP-címekhez fog vezetni.

Ilyeténképpen a forgalom fele-fele arányban megoszlik a két kiszolgáló között, viszont a pinkynek és brainnek most már egymás állapotát is felügyelni kell. Ezt egymás IP-címének pingelésével, és a kapott eredmények ellenőrzésével tudják megtenni. Héjprogram formában mentsük az alábbi programot és telepítsük lemezre:

### Lista: takeover

```
#!/bin/bash
OTHER="brain"
PUBLIC="208.201.239.37"

PAUSE=3

PATH=/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/sbin
MISSED=0

while true; do
if ! ping -c 1 -w 1 $OTHER > /dev/null; then
((MISSED++))
else
if [ $MISSED -gt 2 ]; then
ifconfig eth0:$OTHER down
fi
MISSED=0
fi;

if [ $MISSED -eq 2 ]; then
ifconfig eth0:$OTHER $PUBLIC
#
# ...but see discussion below...
#
fi
sleep $PAUSE;
done
```

Természetesen a brain nevű kiszolgálón futó programban az `OTHER` változót pinkyre, a `PUBLIC`-ot pedig 208.201.239.36-ra kell beállítani.

Tételezzük fel, hogy a brain váratlanul nem válaszol a 208.201.239.17-nek, mert mondjuk, az egyik karbantartó véletlenül a rossz dugót húzta ki kábelszekrényben. Ha három egymás utáni `ping` válasz nélkül marad, akkor pinky működésbe lép, felélesztve az `eth0:brain`-t, a 208.201.239.37-est címet, azt a nyilvános IP-címet, amely a braint szolgáltatja. Ezután folytatja brain valódi címének figyelését, és megszünteti a vezérlést, amint az újra hozzáférhetővé válik. A `ping -c 1 -w -1` parancs azt jelenti, hogy „küldj egy csomagot és egy időtúllé-

pés-jelet (time-out) egy másodperc múlva, bármi is történjék". A `ping` parancs nullától eltérő értéket ad vissza, ha a csomag nem érkezett vissza az 1 másodperces időtúllépésen belül.

Ámde ez még korántsem a teljes megoldás! Annak ellenére, hogy pinky szépen válaszolgat brain kéréseire, a két kiszolgálóval megegyező hálózaton levő gépek, nevezetesen az internet-szolgáltató felé eső útválasztó, hibás MAC-címeket fog a gyorstárba (cache) helyezni a 208.201.239.37-esnek. A gyorstárba helyezett hibás MAC-címekkel semmilyen forgalom nem fog a pinky felé irányulni, mivel az csak olyan csomagokra fog válaszolni, amelyek az ő saját MAC-címét viselik. Hogyan tudjuk a 208.201.239.0-s hálózaton levő összes géppel közölni, hogy a 208.201.239.37-es gép számára a MAC-cím frissítve lett?

Az egyik lehetséges megoldás a High Availability Linux *send_arp* nevű segédprogramja. Ez a könnyen kezelhető és apró segédprogram a meghatározásainknak megfelelő ARP-csomagot állít elő és továbbítja a helyi hálózaton az általunk meghatározott MAC-címre. Ha csupa 1-esből – vagyis ff:ff:ff:ff:ff:ff – álló címet adunk meg, akkor ez tulajdonképpen csoportszórásos ARP-csomaggá válik. Miközben az útválasztók többsége nem szokta rendszeresen frissíteni az ARP-tábláját, az olyan kéretlen csoportszórásos ARP-csomagokat észreveszik, amilyen az ARP-kérelem újraküldését jelző csomag, és erre pinky már kötelező jelleggel válaszolni fog. A csoportszórásos üzenet előnye abban áll, hogy az alhálózaton jelenlevő összes gépnek egyidejűleg küld jelet ahelyett, hogy nyomon kellene követni a gépek MAC-címváltozásainak frissítését.

A `send_arp` parancs írásmódja: `send_arp [forrás IP] [forrás MAC] [cél IP] [cél MAC]`. Például egyszerű, fenti felügyelő héjprogramunknak az alábbi parancsot kellene futtatnia, amikor érzékeli a brain kiszolgáló leállását:

```
send_arp 208.201.239.37 00:11:22:aa:bb:cc
➥ 208.201.239.37 ffffffffffff
```

Ebben a 00:11:22:aa:bb:cc jelsorozat a pinky nevű kiszolgáló `eth0` hálózati csatolójának MAC-címét jelöli. A héjprogram továbbra is képes figyelni, hogy a brain kiszolgáló valódi IP-címe – 208.201.239.17– mikor válik megint elérhetővé. Amint megint hozzáférhető a brain valódi címe, újra lekapcsolhatjuk az `eth0:brain`-t, és rábízhatjuk a brainre az ARP-gyorstára frissítését (például ez elvégezhető a rendszer indulásakor).

Számos lehetőségünk nyílik, hogy továbbfejlesszük ezt a módszert, például a 208.201.239.17 működése nem feltétlenül jelenti azt, hogy a 208.201.239.37 is elérhető. Ezenkívül a `ping` sem a legjobb eszköz a szolgáltatás elérhetőségének ellenőrzésére (ennél jobb lenne a másik kiszolgálóról weboldal kérése, és annak ellenőrzése, hogy a kért oldalnak megvan-e a lezáró `</html>` címkéje).

Ezek a fejlesztések, gyakorlatképpen, az olvasóra várnak. Minden webhely különböző, így magunknak kell megtalálnunk azt módszert, amely a legjobban működik a rendelkezésünkre álló eszközökkel.

### Kapcsolódó anyagok

- A High Availability Linux Fake csomagja: `http://www.linux-ha.org/failover/`
- [#79] Terhelésmegosztás címforgatásos DNS segítségével

## #64 Az ntop futtatása valós idejű hálózati statisztikák készítéséhez

### Nézzünk utána az ntop-pal, ki, mit csinál a hálózatunkon

Ha valós idejű hálózati statisztikákra van szükségünk, okvetlenül vegyük számításba a nagyszerű `ntop` eszközt. Az `ntop` teljes körű szolgáltatást nyújtó webes felületű protokollelemző eszköz, amelyet az SSL-támogatás és a grafikonos ábrázolási lehetőség tesz teljessé. Az `ntop` nem tartozik a pehelysúlyú alkalmazások közé, minthogy a hálózat méretével és a lebonyolított hálózati adatforgalommal együtt növekszik erőforrásigénye, de képes nagyon jó áttekintést adni (és a kényes (critical) részeket kiemelni) arról, ki kivel forgalmaz a hálózaton.

Az ntop-nak először rendszergazdai jogosultsággal kell futnia, hogy a hálózati csatolófelületeket vegyes, azaz csomagokat nem válogató üzemmódba kapcsolja és elindítsa a csomagok fogadását, de ezt követően az előjogokat átruházza arra a felhasználóra, akit kijelölünk. Ha az ntop hosszú időtartamú futtatása mellett döntünk, akkor legjobb lesz erre a célra egy külön gépet biztosítani, amelyen biztonsági és teljesítményi szempontoknak megfelelően kevés más szolgáltatást indítunk.

Az alábbiakban egy rövid hivatkozáslistát adunk közre, hogyan kell az ntop-ot gyorsan üzembe állítani és működtetni. Először hozzuk létre az ntop-felhasználót és csoportot.

```
root@gemini:~# groupadd ntop
root@gemini:~# useradd -c "ntop user"
➥ -d /usr/local/etc/ntop \
  -s /bin/true -g ntop ntop
```

Aztán csomagoljuk ki és építsük össze a *docs/BUILD-NTOP* dokumentumban leírt utasításoknak megfelelően. Azt fogjuk feltételezni, hogy a forrásfát a */usr/local/ntop-2.1.3/* helyre bontotta ki.

Hozzuk létre azt a könyvtárat az ntop számára, ahová a begyűjtött adatokat tartalmazó adatbázist kell tennie.

```
root@gemini:~# mkdir /usr/local/etc/ntop
```

Ne feledjük, hogy tulajdonosnak a rendszergazdát kell beállítani, nem pedig az ntop-felhasználót.

Ha a szabványos HTTP helyett az SSL-lel kiegészített HTTPS-t szeretnénk használni, akkor az alapértelmezés szerinti SSL-kulcsot másoljuk a */usr/local/etc/ntop* állományba:

```
root@gemini:# cp /usr/local/src/ntop-2.1.3/
➥ ntop/*pem /usr/local/etc/ntop
```

Ne feledjük, hogy az alapértelmezés szerinti SSL-kulcs a helyes gépnévvel nem lesz készen a kiszolgáló számára. Ha szeretnénk elkészíteni (és eltüntetni a böngészőprogramban megjelenő figyelmeztető üzeneteket) tanulmányozzuk a [#93] „SSL bizonyítvány készítése és a bizonyítvány aláírási kérelem" című trükköt az SSL-tanúsítvány létrehozásáról. Továbbá a [#94] „Készítsünk saját hitelesítés-szolgáltatót" trükk elolvasása után megtudhatjuk hogyan kell saját azonosító kulcsot létrehozni, és milyen módon kívánatos azzal a hitelesítési tanúsítványt (CA) aláírni.

Ezután el kell végeznünk az `ntop`-adatbázis kezdeti feltöltését és beállítjuk a hozzátartozó rendszergazdai jelszót:

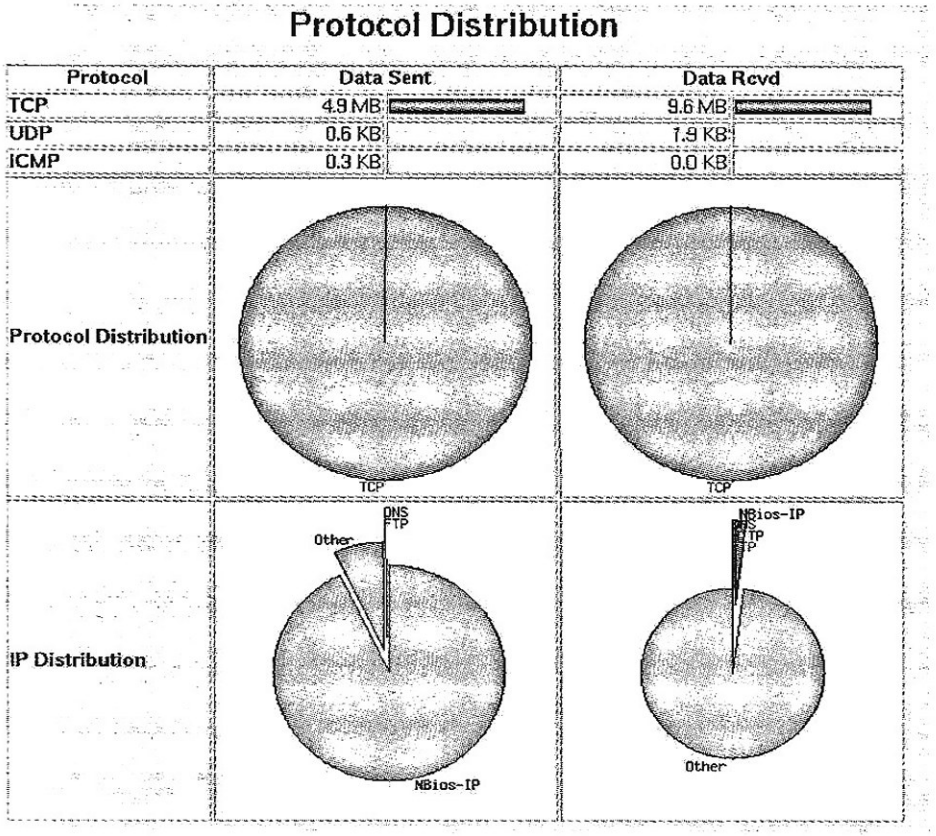
```
root@gemini:~# ntop -A -u ntop
➥ -P /usr/local/etc/ntop
21/Sep/2002 20:30:23 Initializing GDBM...
21/Sep/2002 20:30:23 Started thread (1026)
➥ for network packet analyser.
21/Sep/2002 20:30:23 Started thread (2051)
➥ for idle hosts detection.
21/Sep/2002 20:30:23 Started thread (3076)
➥ for DNS address resolution.
21/Sep/2002 20:30:23 Started thread (4101)
➥ for address purge.

Please enter the password for the admin user:
Please enter the password again:
21/Sep/2002 20:30:29 Admin user password
➥ has been set.
```

Végül futtassuk démonként az `ntop`-ot, és indítsuk el az SSL-kiszolgálót a kedvenc kapunkon, mondjuk a 4242-esen:

```
root@gemini:~# ntop -u ntop
➥ -P /usr/local/etc/ntop -W4242 -d
```

Alapértelmezés szerint az `ntop` szintén szabályos HTTP-kiszolgálóként fut a 3000-es kapun. Erősen fontolóra kell venni e kapuk tűzfalon való lezárását, vagy parancssori `iptables`-szabályok használatát, amint azt

a #45 „Tűzfal létrehozása egy tetszőleges kiszolgáló parancssorából" című trükknél láthattuk. Engedjük futni a kiszolgálót egy ideig, aztán kapcsolódjunk a *https://itt.az.en.kiszolgalom.hu:4242/* felületen a géphez, és itt mindenféle részletet megtalálunk arról, milyen forgalom zajlott a hálózaton.

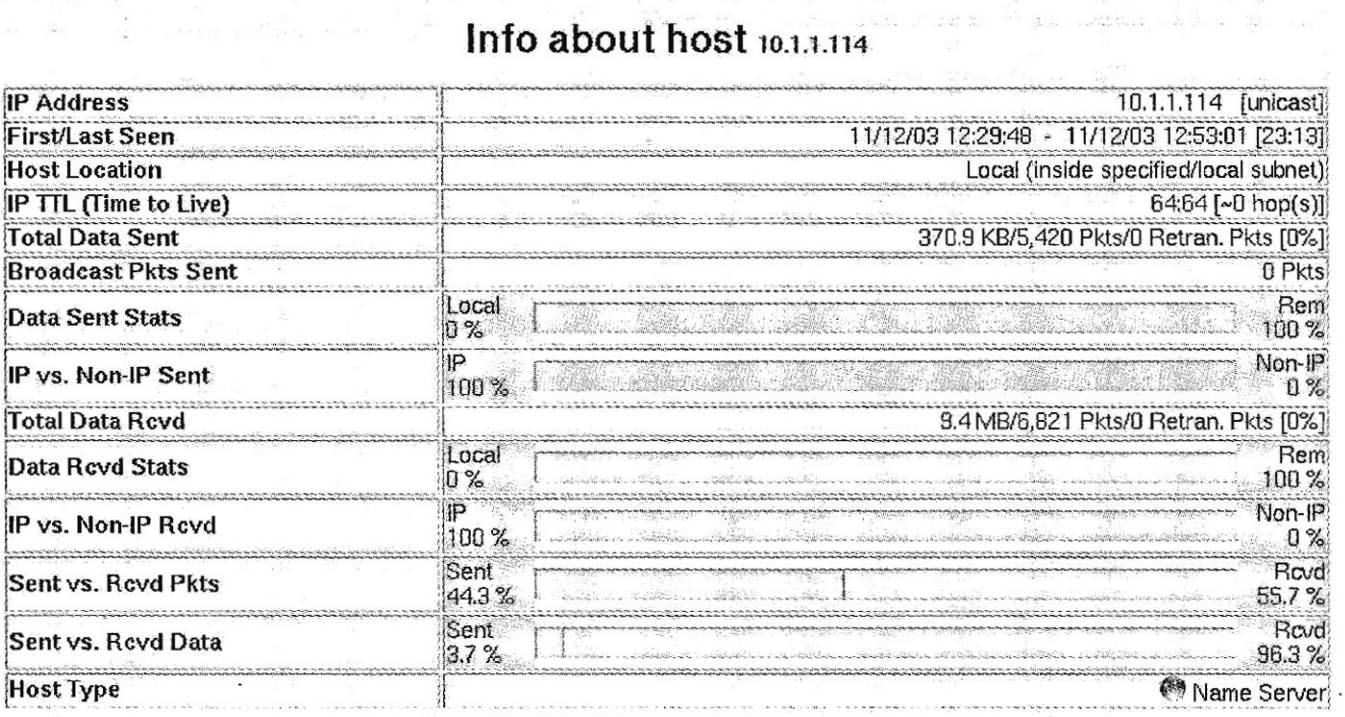



## 5.1. ábra

*Az ntop valós idejű statisztikai grafikonokat készít a hálózatunkon zajló adatforgalomról*

Különböző grafikus statisztikákat láthatunk: könnyen értelmezhető és kinyomtatva is mutatós körgrafikonokat kapunk – amint az 5.2. ábrán is látható.

A `tcpdump` és az `ethereal` a hálózati forgalom részletes és párbeszédes elemzését adja, az `ntop` ezzel szemben statisztikai adatok garmadáját nyújtja könnyen áttekinthető felületen. Ha megfelelően lett telepítve és idegen kezek nem férnek hozzá, könnyen válhat a hálózati rendszerelemző-készlet legkedvesebb darabjává.



**Info about host** 10.1.1.114

| | |
|---|---|
| IP Address | 10.1.1.114 [unicast] |
| First/Last Seen | 11/12/03 12:29:48 - 11/12/03 12:53:01 [23:13] |
| Host Location | Local (inside specified/local subnet) |
| IP TTL (Time to Live) | 64:64 [~0 hop(s)] |
| Total Data Sent | 370.9 KB/5,420 Pkts/0 Retran. Pkts [0%] |
| Broadcast Pkts Sent | 0 Pkts |
| Data Sent Stats | Local 0 % — Rem 100 % |
| IP vs. Non-IP Sent | IP 100 % — Non-IP 0 % |
| Total Data Rcvd | 9.4 MB/6,821 Pkts/0 Retran. Pkts [0%] |
| Data Rcvd Stats | Local 0 % — Rem 100 % |
| IP vs. Non-IP Rcvd | IP 100 % — Non-IP 0 % |
| Sent vs. Rcvd Pkts | Sent 44.3 % — Rcvd 55.7 % |
| Sent vs. Rcvd Data | Sent 3.7 % — Rcvd 96.3 % |
| Host Type | Name Server |

5.2. ábra

*A hosszan tartó programfuttatás során az ntop nagy részletességgel képes megvilágítani, hogyan használják az egyes ügyfelek a hálózatot*

### Kapcsolódó anyagok

- `http://www.ntop.org/`
- `http://www-serra.unipi.it/~ntop/ntop.html`

# #65 Webforgalom valós idejű felügyelete a httptop programmal

## Nézzük meg a httptop-pal, éppen kik használják webkiszolgálónkat

Haladó szintű naplóelemző csomagok, mint például az *analog*, nagyon részletes jelentést képesek készíteni arról, ki mit nézett meg a webhelyünkön. A napló feldolgozása egy forgalmas helyen sajnos sok időt vehet igénybe, s ugyan részletes visszatekintő jelentést ad, de semmit sem tudunk arról, hogy az adott pillanatban mi történik.

Egy másik megközelítés azt mondja, hogy miden idevágó webműveletet adatbázisban kell rögzíteni, és a valós idejű statisztikákhoz le-

kérdezéseket kell végezni. Ez a módszer azonnali, a legfontosabb jellemzőket mutató jelentéssel képes szolgálni, feltéve, hogy az adatbázist és webkiszolgálót befogadó számítógép egy olyan erőmű, amely az üzemeltetéssel jelentkező járulékos terhet is képes elviselni. Nagy forgalmú kiszolgálón azonban ez nem túl szerencsés megoldás.

A megoldást képező `httptop` valahol félúton található e két megoldás között. Ez egy Perl nyelvű héjprogram, amely a *top*-hoz hasonló valós idejű jelentéseket készít, az adatbázis-használattal járó további teher nélkül. Mindössze egy *access_log* naplóállományhoz szükséges elérési utat kell neki megadni. A parancs futtatása valahogy így fest:

```
$ httptop -f combined
➥ /usr/local/apache/logs/access_log
```

Ennek eredményeképpen egy néhány másodpercenként frissülő képernyőt fogunk látni, amelynek rendezése másodpercenként a csúcsértékekre nézve fog megtörténni. A *top*-nál megszokott módon a ? (kérdőjel) a használható parancsok listáját mutatja meg. Például lehet a legfrissebb csúcsértékek vagy összesített csúcsértékek szerinti rendezést kérni, választható a hivatkozó webhely vagy névtartomány megjelenítése, kért cím vagy távoli gépre kérelem megjelenítése.

Ha több VirtualHostot üzemeltetünk és a `httptop`-ot egyidejűleg szeretnénk futtatni rajtuk, akkor próbáljuk ki, hogy minden egyes VirtualHost-bejegyzést a következő sorral egészítsünk ki:

```
CustomLog /usr/local/apache/logs/combined-log vhost
```

Majd a következő egysoros napló-meghatározást el kell helyezni valahol az általános beállításoknál:

```
LogFormat "%v %h %l %u %t \ "%r\ " %>s %b \
➥ "%{ Referer} i\ " \ "%{ User-Agent} i\ "" vhost
```

Most már az összes gépen egyidejűleg figyelhetjük meg a tevékenységet a következő paranccsal:

```
$ httptop -f vhost /usr/local/apache/logs/combined-log
```

Ne feledjük, hogy ez a `httptop` a `Time::HiRes` és `File::Tail` Perl-modulok előzetes telepítését igényli.

### Lista: httptop

```
#!/usr/bin/perl -w
#

=head1 NAME

httptop - display top(1)-like per-client
➥ HTTP access stats

=head1 SYNOPSIS

httptop [-f <format>] [-r <refresh_secs>]
➥ [-b <backtrack_lines>] <logdir | path_to_log>

=cut

use Time::HiRes qw( time );
use File::Tail ( );
use Term::ReadKey;
use Getopt::Std;

use strict;

### Defaults you might be interested in adjusting.

my $Update = 2; # update every n secs
my $Backtrack = 250;
➥ # backtrack n lines on startup
my @Paths = qw(
%
/title/%/logs/access_log
/var/log/httpd/%/access_log
/usr/local/apache/logs/%/access_log
);

my $Log_Format = "combined";
my %Log_Fields = (
combined => [qw/ Host x x Time
➥ URI Response x Referer Client /],
```

```
vhost => [qw/ VHost Host x x Time
➥ URI Response x Referer Client /]
);

### Constants & other thingies.
    ➥ Nothing to see here. Move along.

my $Version = "0.4.1";

sub by_hits_per (  ) {  $b->{ Rate}
➥ <=> $a->{ Rate}  }
sub by_total (  ) {  $b->{ Total}
➥ <=> $a->{ Total}  }
sub by_age (  ) {  $a->{ Last}
➥ <=> $b->{ Last}  }

my $last_field = "Client";
my $index = "Host";
my $show_help = 0;

my $order = \ &by_hits_per;
my $Help = "htlwufd?q";
my %Keys = (
h => [ "Order by hits/second"
➥ => sub {  $order = \ &by_hits_per }  ],
t => [ "Order by total recorded hits"
➥ => sub {  $order = \ &by_total }  ],
l => [ "Order by most recent hits"
➥ => sub {  $order = \ &by_age }  ],
w => [ "Show remote host"
➥ => sub {  $index = "Host" }  ],
u => [ "Show requested URI"
➥ => sub {  $index = "URI" }  ],
f => [ "Show referring URL"
➥ => sub {  $index = "Referer" }  ],
d => [ "Show referring domain"
➥ => sub {  $index = "Domain" }  ],
'?' => [ "Help (this thing here)"
➥ => sub {  $show_help++ }  ],
q => [ "Quit" => sub {  exit }  ]
);
```

```
my @Display_Fields = qw/
➥ Host Date URI Response Client Referer Domain /;
my @Record_Fields = qw/ Host URI Referer Domain /;
my $Max_Index_Width = 50;
my $Initial_TTL = 50;

my @Months = qw/
➥ Jan Feb Mar Apr May Jun Jul Aug Sep Nov Dec /;
my %Term = (
HOME => "\ 033[H",
CLS => "\ 033[2J",
START_TITLE => "\ 033]0;", # for xterms etc.
END_TITLE => "\ 007",
START_RV => "\ 033[7m",
END_RV => "\ 033[m"
);

my ( %hist, %opt, $spec );

$SIG{ INT}  = sub {  exit } ;
END {  ReadMode 0 } ;

### Subs.

sub refresh_output
{
my ( $cols, $rows ) = GetTerminalSize;
my $show = $rows - 3;
my $count = $show;
my $now = (shift || time);

for my $type ( values %hist ) {
for my $peer ( values %$type ) {
# if ( --$peer->{ _Ttl}  > 0 ) {
my $delta = $now - $peer->{ Start} ;
if ( $delta >= 1 ) {
$peer->{  Rate }  = $peer->{  Total }  / $delta;
}  else {
$peer->{  Rate }  = 0
}

$peer->{  Last }  =
➥ int( $now - $peer->{  Date }  );
```

```
# }  else {
# delete $type->{ $peer}
# }
}
}

$count = scalar( values %{ $hist{ $index} }  ) - 1
➥ if $show >= scalar values %{ $hist{ $index} } ;
my @list = ( sort $order values
➥ %{ $hist{ $index} }  )[ 0 .. $count ];

my $first = 0;
$first = ( $first <= $_ ? $_ + 1 : $first )
➥ for map {  $_ ? length($_->{ $index} ) : 0 }
➥ @list;
$first = $Max_Index_Width
➥ if $Max_Index_Width < $first;

print $Term{ START_TITLE} , "Monitoring $spec at: ",
➥ scalar localtime, $Term{ END_TITLE}
➥ if $ENV{ TERM}  eq "xterm"; # UGLY!!!

my $help = "Help/?";
my $head = sprintf( "%-${ first}
➥ s %6s %4s %4s %s (%d total)",
$index, qw{  Hits/s Tot Last } , $last_field,
scalar keys %{ $hist{ $index} }
);

#
# Truncate status line if need be
#
$head = substr($head, 0, ($cols - length($help)));
print @Term{ "HOME", "START_RV"} ,
➥ $head, " " x ($cols - length($head)
➥ - length($help)), $help, $Term{ END_RV} , "\ n";

for ( @list ) {
# $_->{ _Ttl} ++;

my $line = sprintf( "%-${ first}
➥ s %6.3f %4d %3d %s",
```

```
substr( $_->{ $index} , 0, $Max_Index_Width ),
➥ @$_{ (qw{  Rate Total Last } , $last_field)}  );
if ( length($line) > $cols ) {
substr( $line, $cols - 1 ) = "";
}  else {
$line .= " " x ($cols - length($line));
}
print $line, "\ n";
}

print " " x $cols, "\ n" while $count++ < $show;
}

sub process_line
{
my $line = shift;
my $now = ( shift || time );
my %hit;

chomp $line;
@hit{ @{ $Log_Fields{ $Log_Format} } }
➥ = grep( $_, split( /"([^"]+)"|\
➥ [([^]]+)\ ]|\ s/o, $line ) );

$hit{  URI }  =~ s/HTTP\ /1\ S+//gos;

$hit{  Referer }  = "<unknown>"
➥ if not $hit{ Referer}  or $hit{ Referer}
➥ eq "-";
( $hit{ Domain}  = $hit{ Referer}  ) =~
➥ s#^\ w+://([^/]+).*$#$1#os;

$hit{  Client }  ||= "<none>";
$hit{  Client }  =~ s/Mozilla\ /[\ w.]+
➥ \ (compatible; /(/gos;
$hit{  Client }  =~ s/[^\ x20-\ x7f]//gos;

# if $now is negative, try to guess how old
  ➥ the hit is based on the time stamp.
if ( $now < 0 ) {
my @hit_t = ( split( m![:/\ s]!o,
➥ $hit{  Time }  ))[ 0 .. 5 ];
```

```
my @now_t = ( localtime )[ 3, 4, 5, 2, 1, 0 ];
my @mag = ( 3600, 60, 1 );

# If the hit didn't parse right, or didn't happen
  ➥ today, the hell with it.
return unless $hit_t[2] == ( $now_t[2] + 1900 )
and $hit_t[1] eq $Months[ $now_t[1] ]
and $hit_t[0] == $now_t[0];

splice( @hit_t, 0, 3 );
splice( @now_t, 0, 3 );

# Work backward to the UNIX time of the hit.
$now = time;
$now -= (shift( @now_t ) - shift( @hit_t ))
➥ * $_ for ( 3600, 60, 1 );
}

$hit{  Date }  = $now;

for my $field ( @Record_Fields ) {
my $peer = ( $hist{ $field} { $hit{ $field} }  ||=
➥ {  Start => $now, _Ttl => $Initial_TTL }  );
@$peer{  @Display_Fields }  =
➥ @hit{  @Display_Fields } ;
$peer->{  Total } ++;
}
}

sub display_help {
my $msg = "httptop v.$Version";
print @Term{ qw/ HOME CLS START_RV /} ,
➥ $msg, $Term{ END_RV} , "\ n\ n";
print " " x 4, $_, " " x 8, $Keys{ $_} [0],
➥ "\ n" for ( split "", $Help );
print "\ nPress any key to continue.\ n";
}


### Init.

getopt( 'frb' => \ %opt );
```

```
$Backtrack = $opt{ b}  if $opt{ b} ;
$Update = $opt{ r}  if $opt{ r} ;
$Log_Format = $opt{ f}  if $opt{ f} ;
$spec = $ARGV[0];

die <<End unless $spec
➥ and $Log_Fields{ $Log_Format} ;
Usage: $0 [-f <format>] [-r <refresh_secs>]
➥ [-b <backtrack_lines>] <logdir | path_to_log>
Valid formats are: @{ [ join ", ",
➥ keys %Log_Fields ]} .
End

for ( @Paths ) {
last if -r $spec;
( $spec = $_ ) =~ s/%/$ARGV[0]/gos;
}
die "No access_log $ARGV[0] found.\ n"
➥ unless -r $spec;

my $file = File::Tail->new(
name => $spec,
interval => $Update / 2,
maxinterval => $Update,
tail => $Backtrack,
nowait => 1
) or die "$spec: $!";

my $last_update = time;
my ( $line, $now );

# Backtracking.
while ( $Backtrack-- > 0 ) {
last unless $line = $file->read;
process_line( $line, -1 );
}
$file->nowait( 0 );

ReadMode 4; # Echo off.
print @Term{ "HOME", "CLS"} ; # Home & clear.
refresh_output;
```

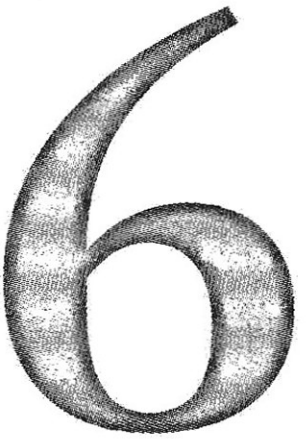
```
### Main loop.

while (defined( $line = $file->read )) {
$now = time;

process_line( $line, $now );

while ( $line = lc ReadKey(-1) ) {
$show_help = 0 if $show_help;
$Keys{ $line} [1]->(  ) if $Keys{ $line} ;
}

if ( $show_help == 1 ) {
display_help;
$show_help++; # Don't display help again.
}  elsif ( $now - $last_update >
➥ $Update and not $show_help ) {
$last_update = $now;
refresh_output( $now );
}
}

__END__

=head1 DESCRIPTION
```

A `httptop` programot `top`-megfelelőnek szánták a `httpd`-elérés nyomon követésére.

A `httptop`-ot Apache kiszolgálóhoz tartozó *access_log* állományhoz tartozó elérési útvonal megadásával kell indítani. A keresési útvonalakat már a forráskódban be lehet állítani.

A `httptop` a különböző naplóformátumok kezelésében csak korlátozott lehetőségekkel bír. Először futtassuk a `httptop`-ot kapcsolók nélkül, hogy megtudjuk, tulajdonképpen milyen formátumok hozzáférhetők.

A `httptop` futása közben a ? (kérdőjel) lenyomásával megkaphatjuk a használható parancsok listáját. Amint azt már a top(1)-nél megszokhattuk, és a `Q` lenyomására a program végrehajtása befejeződik.



# Az SSH

## 66–71. trükkök

Miközben a különféle trükköket gyűjtögettem a könyvhöz, hamar világossá vált számomra, hogy az *SSH* külön fejezetet érdemel. Az SSH olyan rugalmas és biztonságos titkosítási módszert nyújt számunkra, amellyel hálózaton keresztül összekapcsolhatjuk a különféle gépek közötti adatfolyamokat. A parancssor Linux alatt lényegében adatok olvasását (és írását) jelenti fájlokból és csővezetékekből. Az SSH lehetővé teszi számunkra, hogy az adatot többé-kevésbé úgy dobjuk keresztül a hálózaton, mintha csak egyetlen gépen dolgoznánk. Mindezt gyors, biztonságos és logikus módon tehetjük meg, ráadásul néhány igen érdekes (és hatékony) trükköt is alkalmazhatunk.

Linux alatt több SSH-változat között is válogathatunk. E fejezet példáiban feltételezzük majd, hogy az OpenSSH v3.4p1 vagy ennél újabb változatát használjuk. Az OpenSSH az összes nagyobb Linux-terjesztésben megtalálható, illetve a `http://www.openssh.com/` címről tölthető le.

# #66 Gyors bejelentkezés SSH-ügyfélkulcsokkal

## Ha jelszavak helyett SSH-kulcsokat használunk, felgyorsíthatjuk és önműködővé tehetjük a bejelentkezést

Amikor egynél több gépen látunk el rendszergazdai feladatokat, létfontosságú, hogy képesek legyünk gyorsan előhívni bármely kiszolgáló héjprogramját. Ha mindig be kell gépelnünk az `ssh en.kiszolgalom.hu` alakot (majd a jelszót), az nem csak fárasztó, de a figyelmünket is megtöri. Nem egy rendszergazdánál jelentkeztek a korai szenilitás jelei, amikor hirtelen át kellett váltania a „Mi is a probléma?" feladatról a „Hogyan jutok oda?" megoldására, amit a „Hol is tartottam?" követett. Ez a „Miért is jöttem be ebbe a szobába?" digitális változata. (Ráadásul a */usr/games/fortune* jelenléte csak fokozza a gondokat!)

Akárhogy is vesszük, minél több energiát fektetünk a bejelentkezésbe, annál kevesebb jut a feladat megoldására. Az SSH mostanában megjelent változatai a vég nélküli jelszógépelgetés helyett új lehetőséget kínálnak: a nyílt kulcsok cseréjét.

Ha nyílt kulcsokat akarunk használni az SSH-kiszolgálón, először is létre kell hoznunk egy nyilvános, illetve saját kulcspárt:

```
$ ssh-keygen -t rsa
```

Használhatjuk `-t dsa` kapcsolót is, ha DSA-kulcsokat szeretnénk, vagy a `-t rsa1` kapcsolót, ha ragaszkodunk a Protocol v1 rendszerhez. (Bár az ilyesmi szégyen és gyalázat, amint lehet, váltsunk v2-re!) Miután begépeltük a fenti parancsot, valami ilyesmit kell látnunk:

```
Generating public/private rsa key pair.
Enter file in which to save the key
➥ (/home/rob/.ssh/id_rsa):
```

Itt egyszerűen csak nyomjunk ENTERT. Ezután a rendszer egy jelmondatot kér tőlünk; nyugodtan üssünk két ENTERT (természetesen csak miután elolvastuk az alatta megjelenő biztonsági figyelmeztetést). Az eredmény a következőképpen fog festeni:

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in
➥ /home/rob/.ssh/id_rsa.
Your public key has been saved in
➥ /home/rob/.ssh/id_rsa.pub.
The key fingerprint is:
a6:5c:c3:eb:18:94:0b:06:a1:a6:29:58:fa:80:0a:bc
➥ rob@localhost
```

Munkánk gyümölcseként két fájl jön létre *~/.ssh/id_rsa* és *~/.ssh/id_rsa.pub* néven. Ha valamelyik kiszolgálón ezt a kulcspárt szeretnénk felhasználni, a következőket írjuk be:

```
$ ssh server "mkdir .ssh; chmod 0700 .ssh"
$ scp .ssh/id_rsa.pub server:.ssh/authorized_keys2
```

Természetesen a *kiszolgáló* helyére a saját kiszolgálónk neve kerül. Mindkét esetben jelszót kell kérnie tőlünk. Ezt követően egyszerűen próbáljuk ki az `ssh` *`kiszolgáló`* parancsot, és ha minden jól megy, önműködően jelszó nélkül léphetünk be a kiszolgálóra. Igen, ezentúl az `scp` is az új, frissen csillogó nyilvános kulcsunkat fogja használni.

Amennyiben valami nem működik, ellenőrizzük a *~/.ssh/** és a *kiszolgáló:~/.ssh/** könyvtárak jogosultságait. Saját kulcsunk (`id_rsa`) jogosultsága 0600 legyen (és csak a saját gépünkön szabad léteznie), minden másnak 0655 vagy ennél erősebb jogosultságokkal kell rendelkeznie.

Nagyszerű! Mostantól könnyedén, csekély kínlódással léphetünk be SSH-val a kiszolgálóra. Létezik olyan módszer, amellyel még ennél is gyorsabban léphetünk be a gyakran használt gépekre? Mérget vehetünk rá. Csak bele kell nézni a [#67] „Villámgyors SSH-bejelentkezés" trükkbe.

### Biztonsági megfontolások

Néhányan úgy vélik a nyilvános kulcsok alkalmazása lehetséges biztonsági rés. Végső soron csak a saját kulcsot kell ellopnia valakinek, és máris hozzáférhet kiszolgálóinkhoz. Bár ez kétségtelenül igaz, de ugyanez nyilvánvalóan elmondható a jelszavakról is.

Kérdezzük csak meg magunktól, naponta hányszor gépelünk be jelszavakat, hogy elérjünk egy gépet (vagy `scp`-vel másoljunk át valamilyen fájlt)? Milyen gyakori, hogy ugyanazt a jelszót használjuk sok (vagy akár az összes) gépen? Használtuk már ezt a jelszót olyan helyen, aminek a biztonsága megkérdőjelezhető (weboldalon, saját gépen, mely nem feltétlenül naprakész, esetleg egy olyan gép SSH-ügyfelén, amelyet nem mi kezelünk)? Ha az előbbi lehetőségek közül bármelyik is ismerősen cseng, nyugodtak lehetünk, mert ugyanilyen feltételek mellett az SSH-kulcs használata is lehetetlenné teszi a támadó számára, hogy jogosulatlan elérést szerezzen (feltéve, hogy a saját kulcsunkat titokban tudjuk tartani).

### Kapcsolódó anyagok

- *SSH: The Definitive Guide* (O'Reilly)
- [#67] Villámgyors SSH-bejelentkezés
- [#69] Az SSH-Agent futtatása grafikus felületen

## #67 Villámgyors SSH-bejelentkezés

### Még gyorsabb bejelentkezés parancssorból

Aki az előző trükkben leírt módosítást elvégezte, nem árt, ha tudja még csak a megoldás felét látta. Az ügyfélkulcsok használatával még mindig feleslegesen kell begépelnünk az `ssh` kiszolgálóparancsot valahányszor SSH-val lépünk be valahová. Még a sötét, biztonságot nélkülöző, megvilágosodás előtti *rsh*-s időkben létezett egy különös képesség, amit nagyon kedveltem, és amit (egyelőre még) nem vittek át SSH alá. Akkoriban ugyanis lehetőség nyílt arra, hogy ha a kiszolgáló nevével azonos elnevezésű közvetett hivatkozásokat hoztunk létre a */usr/bin/rsh* fájlra, az *rsh* elég okos volt és kitalálta, hogy mivel nem *rsh* néven hívták meg, a nevével azonos kiszolgálóra kell belépnie.

Mindezt könnyen utánozhatjuk parancsfájl segítségével. Készítsünk egy parancsfájlt *ssh-to* néven, amely a következő két sort tartalmazza:

```
#!/bin/sh
ssh `basename $0` $*
```

(A `basename $0` körül található jelek fordított egyszeres idézőjelek – backtick –). Az előző fájlt helyezzük az elérési utunkba (feltéve, hogy a *~/bin* nincs már eleve a `PATH` változónkban) és készítsünk közvetett hivatkozásokat kedvenc kiszolgálóink nevével:

```
$ cd bin
$ ln -s ssh-to kiszolgáló1
$ ln -s ssh-to kiszolgáló2
$ ln -s ssh-to kiszolgáló3
```

Mostantól fogva, ha `ssh`-zni szeretnénk *kiszolgáló1*-re (feltételezve, hogy a korábban leírt módszerrel már átmásoltuk nyilvános kulcsunkat), mindössze a kiszolgáló nevét kell beírnunk, és varázslatos módon máris a nevezett kiszolgáló héjprogramjában találjuk magunkat, mindenféle SSH parancsok és jelszavak megadása nélkül. A sor végén található `$*` arra szolgál, hogy (a héjprogram elindítása helyett) egyetlen sorban tetszőleges parancsokat adhassunk ki, valahogy így:

```
kiszolgáló1 uptime
```

(Az előző sor megmutatja *kiszolgáló1* futásidejét, felhasználóinak számát és terhelési átlagát, majd kilép. Ha kíváncsiak vagyunk az összes gépünk állapotára, tegyük `for` ciklusba a parancsot és lépkedjünk végig a kiszolgálók listáján.)

Úgy gondolom, ez az SSH használatának lehető leggyorsabb módja, eltekintve attól az esettől, amikor egykarakteres álneveket készítünk (ami egyébként szélsőséges, nehezen karbantartható és szükségtelen tákolás, ugyanakkor néhány embert láthatóan lebilincsel):

```
$ alias a='ssh alice'
$ alias b='ssh bob'
$ alias e='ssh eve'
...
```

Az SSH számos olyan képességgel bír, amelyek a különböző kiszolgálók közötti navigálás igen rugalmas eszközévé teszik. Sikerült, megtaláltuk a módját miképpen jelentkezzünk be és illesszük a rendszerbe gépeinket a legkönnyebben.

### Kapcsolódó anyagok

- [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal
- [#68] SSH-Agent, hatékonyan

## #68 SSH-Agent, hatékonyan

### Használjuk az SSH-Agent programot SSH-ügyfélkulcsaink önműködő kezeléséhez

Az SSH egyik nagyon hasznos összetevője az *SSH-Agent*, amely helyettünk kezeli saját kulcsainkat és szükség szerint átadja hitelesítő adatainkat. Az *SSH-Agent* súgóoldala a következőket tudatja velünk:

*„Az SSH-Agent (SSH-ügynök) célja, hogy a nyilvános kulcsú kódolásokban (RSH, DSA) használt saját kulcsokat tárolja. Az alapelképzelés szerint az SSH-Agentet az X-folyamat vagy a bejelentkezés előtt indítjuk el, így valamennyi ablak és program az SSH-Agent program ügyfeleként indul el. Az ügynök környezeti változókon keresztül érhető el és az azonosítási folyamat során önműködően felhasználható valahányszor más gépekre jelentkezünk be az SSH(1) segítségével."*

A gyakorlatban ez annyit jelent, hogy a háttérben futó ügynökkel (és egy helyesen beállított SSH-ügyféllel) egymás után több gépre is beléphetünk az SSH-val anélkül, hogy valamennyi köztes gépen létezne egy-egy másolat a saját kulcsunkból (vagy mindenhol jelszót kellene begépelnünk).

Tételezzük fel, hogy létezik egy hitelesített SSH-kulcsunk (lásd a [#66] „Gyors bejelentkezés SSH-ügyféllel" trükköt) a homer, bart és lisa gépeken. Amíg a saját gépünkről `ssh`-zunk ezekre a gépekre, nem találkozunk semmiféle nehézséggel:

```
rob@caligula:~$ ssh homer
rob@homer:~$ exit
logout
Connection to homer.oreillynet.com closed.
rob@caligula:~$ ssh bart
```

```
rob@bart:~$ exit
logout
Connection to bart.oreillynet.com closed.
rob@caligula:~$ ssh lisa
rob@lisa:~$ exit
```

De mi történik akkor, ha a homer gépről közvetlenül `ssh`-zunk át bart-ra?

```
rob@caligula:~$ ssh homer
rob@homer:~$ ssh bart
rob@bart's password:
```

Nos, ilyenkor lép a képbe az *SSH-Agent*. Ahelyett, hogy felesleges veszélyeknek tennénk ki saját kulcsunkat, és az összes kiszolgálóra egy-egy másolatot helyeznénk el belőle, inkább a saját gépünkön indítsuk el az ügynököt:

```
rob@caligula:~$ eval `ssh-agent`
Agent pid 8450
```

Majd alapértelmezett SSH-kulcsainkat vegyük fel az `ssh-add` paranccsal:

```
rob@caligula:~$ ssh-add
Identity added: /home/rob/.ssh/id_rsa
➥ (/home/rob/.ssh/id_rsa)
Identity added: /home/rob/.ssh/id_dsa
➥ (/home/rob/.ssh/id_dsa)
Identity added: /home/rob/.ssh/identity
➥ (rob@caligula)
```

Ellenőrizzük, hogy a homer, bart és lisa gépeken beállítottuk-e az ügynökök kérelmeinek továbbítását. Ez ugyanis általában le van tiltva, de a következő sor beírásával engedélyezhető:

```
ForwardAgent yes
```

A fenti sort a *~/.ssh/config* vagy a */usr/local/etc/ssh_config* fájlba be kell jegyezni. Ezenkívül parancssorból is megadhatjuk a `-A` kapcsoló felhasználásával. Mostantól, ha a homer-ről közvetlenül lépünk be bart-

ra, a homer először az ügynökhöz fordul a szükséges hitelesítő adatokért. Hasonlóképpen, amikor a bart-ról lépünk tovább a lisára, bart előbb homer-hez fordul kérésével, ami azután továbbítja a kérést ügynökünkhöz. Ilyen módon könnyűszerrel ugrálhatunk gépről gépre.

```
rob@caligula:~$ ssh homer
rob@homer:~$ ssh bart
rob@bart:~$ ssh lisa
rob@lisa:~$
```

Gratulálunk, sikerült létrehoznunk egy nagyon egyszerű hálózati navigációs rendszert, melyben nem kell aggódnunk SSH saját kulcsunk biztonsága miatt. Még a gépek közötti másolás (`scp`) is gyorsabb és könnyebb lesz, mint azelőtt volt.

De mi történik, ha nem áll módunkban a legelején elindítani az ügynököt? (Gyakran ez a helyzet, amennyiben grafikus XDM bejelentkező felületen lépünk be, vagy ha OS X-et futtatunk.) Aggodalomra semmi ok, legalábbis, amíg nem olvastuk a [#69] „Az SSH-Agent futtatása grafikus felületen" című trükköt. Amennyiben fontosnak tartjuk a biztonságot, ennél gyorsabban és egyszerűbben nemigen használhatjuk az SSH-t. Esetleg mégis? A választ a [#67] „Villámgyors SSH-bejelentkezés" című trükkben találjuk.

## #69 Az SSH-Agent futtatása grafikus felületen

### Az SSH-Agent futtatása ablakos környezetben

A [#68] „SSH-Agent, hatékonyan" című trükkben a bejelentkezésre bemutatott tipp addig működik jól, amíg a kezdeti `eval ssh-agent` parancsot még a grafikus felület indulása előtt ki tudjuk adni. Ezt a bejelentkezés során például a *~/.bash_login* (vagy *tcsh* alatt a *~/.login*) fájlban tehetjük meg. Amennyiben jelmondatokat használunk kulcsainkhoz (valószínűleg ezt szeretnénk), ez egyáltalán nem hatékony megoldás. Ablakkezelő rendszerünk mindaddig nem fog elindulni, amíg a bejelentkezés után be nem gépeljük a kulcsok jelszavait. A módszer másik mellékhatása, hogy ha az *SSH-Agent* ügynökprogram valamiért leáll, mindig ki kell lépnünk az X-ből, új ügy-

nököt kell indítanunk, majd újra be kell jelentkeznünk. Ráadásul néhány környezetben (például ilyen az OS X is) egyáltalán nincs is lehetőségünk parancsok futtatására a grafikus környezet indulása előtt.

Ahhoz, hogy ne kelljen elindítanunk feleslegesen minden egyes megnyitott ablakban az *SSH-Agent*-et (vagy átmásolgatni a környezeti változókat), próbáljuk meg a következő kódot beilleszteni *~/.profile* állományunkba:

```
if [ -f ~/.agent.env ]; then
. ~/.agent.env > /dev/null

if ! kill -0 $SSH_AGENT_PID > /dev/null 2>&1 then
echo "Stale agent file found. Spawning new
      ➥ agent..."
eval `ssh-agent | tee ~/.agent.env`
ssh-add
fi
else
echo "Starting ssh-agent..."
eval `ssh-agent | tee ~/.agent.env`
ssh-add
fi
```

A kód készít számunkra egy *~/.agent.env* állományt, amelyben az éppen futó *SSH-Agent*-re mutató környezet található. Amikor az ügynök „meghal", új terminál létrehozásával önműködően újabb indul (és egyben kulcsaink is betöltődnek), amit az összes további ablak már osztottan használ.

Na, ez már jobban hangzik: kívánságra újjáéledő *SSH-Agent.*

## Kapcsolódó anyagok

- [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal
- [#67] Villámgyors SSH-bejelentkezés

# #70 X az SSH fölött

## Futtassunk könnyedén és biztonságosan SSH-n keresztül távoli X11-alkalmazásokat

Meglepően kevesen tudják, hogy az SSH kiválóan alkalmas az X11 adatforgalom továbbítására. Amennyiben olyan SSH-kiszolgálóra jelentkeztünk be, amely engedélyezi az X11 továbbítást, az X-alkalmazások indítása mindössze ennyiből áll:

```
rob@florian:~$ ssh -X catlin
Last login: Thu Sep 5 22:59:25 2002
➥ from florian.rob.nocat
Linux 2.4.18.

rob@catlin:~$ xeyes &
[1] 12478
rob@catlin:~$
```

Amennyiben a helyi gépünkön X fut, a fenti sorok az *xeyes* programot jelenítik meg asztalunkon. Azonban ez az *xeyes* valójában a catlin-on fut, azon gépen, amire éppen most jelentkeztünk be. A teljes X11 forgalom titkosítva utazik az SSH-kapcsolaton keresztül, és végül a helyi gépen jelenik meg.

A valódi munkát az SSH végzi, amely tulajdonképpen egy helyi X11 proxykiszolgálót hoz létre számunkra:

```
rob@catlin:~$ echo $DISPLAY
catlin:10.0
```

Az X11 átirányítás OpenSSH alatt alapértelmezés szerint tiltott. Engedélyezéséhez adjuk a következő sort a *sshd_config* állományunkhoz, majd indítsuk újra az `sshd`-t:

```
X11Forwarding yes
```

Igaz, az *xeyes* talán nem a legjobb példa miért is akarnánk ilyesmit tenni. Jöjjön néhány lehetőség, ami talán hasznosabbnak fog tűnni:

`ethereal`
: Csomagelfogást és képi elemzést végez a kiszolgálón.

`vnc`
: A helyi gépről átveszi a vezérlést a távoli X-munkafelület felett, mintha csak a konzol mellett ülnénk.

`gkrellm`
: Tetszetős grafikus formában mutatja be a kiszolgálórendszer állapotát (egyszerre akár többet is).

Ha az X11 forgalmat önműködően szeretnénk az SSH-n keresztül irányítani, a következő sort írjuk a *~/.ssh/config* állományba:

```
ForwardX11 yes
```

Amennyiben ez a beállítás érvényes, többé nincs szükségünk a `-X` kapcsolóra. E módszert az *ssh-to* parancsfájl-megoldással egyesítve (amint azt a [#66] „Gyors bejelentkezés SSH-ügyfélkulcsokkal" című részben láthattuk) máris nagyon gyors trükkhöz juttatott, amellyel biztonságosan indíthatunk távoli X11-folyamatokat:

```
rob@florian:~$ catlin ethereal &
```

Ez a parancs azonnal elindít egy catlin-on futó SSH-n keresztül kódolt `ethereal` folyamatot. Amennyiben leállítjuk az SSH-folyamatot (mondjuk a `~.` segítségével), az összes alatta futó X-alkalmazás szintén azonnal leáll. De ha csak ideiglenesen akarjuk szüneteltetni őket, használhatjuk az SSH-folyamat felfüggesztését (a `~^Z` segítségével), később pedig visszatérhetünk az `fg` paranccsal.

## Kapcsolódó anyagok

- Az Ethereal csomagfigyelő a `http://www.ethereal.com/` címen érhető el.
- A VNC (ingyenes távoli X-munkafelületügyfél) `http://www.uk.research.att.com/vnc/` címen található.
- A *gkrellm* rendszerfigyelő a `http://www.gkrellm.net/` címről tölthető le.

# #71 Kaputovábbítás az SSH-n keresztül

## Tegyük biztonságossá tetszőleges kapuk hálózati forgalmát SSH kaputovábbítással

A távoli héjprogramelérésen, illetve parancsvégrehajtáson túl, az OpenSSH képes bármely tetszőlegesen választott TCP-kaput a kapcsolat másik oldalára továbbítani. Ez igen hasznos lehet, ha például leveleket, webtartalmat vagy egyéb bizalmasan kezelendő adatot szeretnénk védeni (legalább a kapcsolat másik végéig).

Az SSH a következőképpen végzi a helyi kaputovábbítást: kapcsolódik egy helyi kapuhoz, végrehajtja a titkosítást, a titkosított adatot továbbítja az SSH-kapcsolat másik végére, visszakódolja, majd továbbítja az általunk megadott távoli gép meghatározott kapujára. Egy `ssh` csővezetéket a `-L` kapcsolóval indíthatunk (az l az angol local (helyi) szó rövidítése):

```
root@laptop:~# ssh -f -N
➥ -L110:mailhost:110 -l user mailhost
```

Természetesen a *nevet* a felhasználónevünkkel, a *mailhost*-ot pedig a levélkiszolgáló nevével vagy IP-címével kell helyettesíteni. Megjegyezzük, hogy példánkban a laptopon rendszergazdai jogosultságokkal kell rendelkeznünk, hiszen kitüntetett kapuhoz szeretnénk kapcsolódni (a 110-es kapu a POP-szolgáltatásé). Nem árt, ha egyúttal kikapcsoljuk az esetlegesen futó helyi POP-démonokat is (nézzünk körül a */etc/inetd.conf* fájlban), máskülönben ütközés léphet fel.

Amennyiben a teljes POP-forgalmat titkosítani szeretnénk, levelezőügyfelünket állítsuk be, hogy a helyi gép 110-es kapujára csatlakozzon. A program boldogan fog kapcsolódni a levélkiszolgálóhoz, ugyanúgy mintha közvetlenül tenné, csak éppen ezúttal a teljes üzenetváltás titkosítottan zajlik.

A `-f` háttérbe küldi az SSH-t, a `-N` pedig azt jelenti, hogy semmilyen parancsot nem kell végrehajtani a távoli gépen (csak a továbbítást kell elvégezni). Amennyiben SSH-kiszolgálónk támogatja, próbáljuk meg bekapcsolni a `-C` kapcsolót, amely elindítja a tömörítést. Ezáltal lényegesen mérsékelhető leveleink letöltési ideje.

A kapcsolat létrehozásakor annyi `-L` kapcsolót adhatunk meg, amennyit csak akarunk. Például, ha a kimenő levélforgalmat is át akarjuk irányítani, a következőt írjuk be:

```
root@laptop:~# ssh -f -N -L110:mailhost:110
➥ -L25:mailhost:25 \
  -l user mailhost
```

Állítsuk be kimenő levelező gazdagépnek a helyi gépünket (*localhost*) és a kimenő levelezésünk máris titkosítottan utazik a *mailhost* gépig. Ennek csak akkor van értelme, ha a levél címzettje a belső hálózaton található, illetve ha nem bízhatunk meg a belső kapcsolatunkban (ilyen például a legtöbb vezeték nélküli hálózati kapcsolat). Nyilvánvaló módon levelünk a *mailhost* elhagyása után már kódolás nélkül továbbítódik, hacsak nem titkosítottuk korábban valamilyen egyéb módon, például *pgp* vagy *gpg* segítségével.

Amennyiben már bejelentkeztünk a távoli gépre, és sürgősen kaputovábbítást akarunk végrehajtani a következőket próbáljuk meg:

- nyomjuk le az ENTER-t,
- gépeljük be: ^C,
- ekkor az `ssh>` parancssorba kellett jutnunk; írjuk be a `-L` sort ugyanúgy, mintha parancssorból adtuk volna meg. Például:

```
rob@catlin:~$
rob@catlin:~$ ~, then C (it doesn't echo)
ssh> -L8080:localhost:80
Forwarding port.
```

Héjprogarmunk ettől kezdve a 8080-as kaput a *catlin* 80-as kapujára fogja továbbítani, éppúgy mintha már elsőre is így adtuk volna meg.

A `-g` kapcsoló használatával azt is megengedhetjük, hogy más (távoli) ügyfelek kapcsolódjanak a továbbított kapunkhoz. Ha például bejelentkezünk a helyi hálózat NAT-szolgáltatást nyújtó távoli átjárójára, a következő parancsot kell kiadnunk:

```
rob@gateway:~$ ssh -f -g -N
➥ -L8000:localhost:80 10.42.4.6
```

Az átjáró 8000-es kapuját célzó valamennyi kapcsolat a 10.42.4.6-os számú belső gép 80-as kapujára továbbítódik. Amennyiben az átjáró élő webcímmel bír, a hálózaton bárki kapcsolódhat a 10.42.4.6-os gép webkiszolgálójára, mintha az átjáró 8000-es kapuján futna.

Még egy dolgot érdemes megemlíteni: az átirányított gépnek nem kötelező helyi gépnek lennie. Bármilyen gép lehet, amit a kapcsolat végén álló gép közvetlenül el tud érni. Például, ha a helyi 5150-es kaput akarjuk a belső hálózat valamelyik webkiszolgálójára továbbítani, a következőt gépeljük be:

```
rob@remote:~$ ssh -f -N
➥ -L5150:intranet.insider.nocat:80 gateway.nocat.net
```

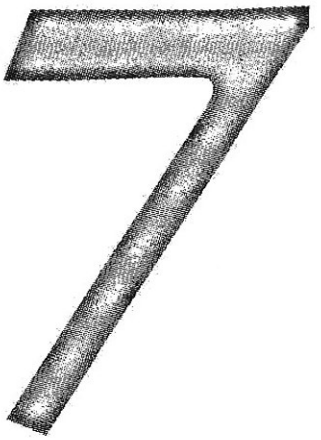
Feltételezve, hogy a *.nocat* nevű TLD-t használjuk (lásd [#80] Saját felsőszintű tartomány üzemeltetése), illetve, hogy a *gateway.nocat.net*-ről el lehet érni a *.nocat* belső hálózatot, a remote nevű gép 5150-es kapujára érkező forgalom engedelmesen továbbítódik az *intranet.insider.nocat* 80-as kapujára. A remote gépnek nem kell tudni kikeresni a DNS-ből az *intranet.insider.nocat* címet. A feloldásra csak akkor kerül sor, amikor a kapcsolat létrejött a *gateway.local.net*-tel, és így az átjáró fogja végrehajtani. Ha a remote-ról biztonságosan akarjuk böngészni az oldalt, kapcsolódjunk a *http://localhost:5150/* címre.

Bár az SSH Socks 4 proxyként is képes működni (a `-D` kapcsoló segítségével), nem túl jó megoldás az összes adatot a kapcsolat másik oldalára továbbítani. Vessünk egy pillantást az [#50] „Alagutazás: IP-IP tokozás" című trükkre, melyből megtudhatjuk hogyan használhatjuk a `vtun` eszközt és az SSH-t az összes adat átirányítására. Olvassuk el a `-D` kapcsoló leírását; nagyon hasznos kis lehetőség. (Csak nem gondolod, hogy mindent megcsinálunk helyetted?)

Az SSH elképesztően rugalmas eszköz, messze több szolgáltatással, mint amit itt le tudtunk írni. A következő hivatkozások közt további SSH-val kapcsolatos érdekességekre bukkanhatunk.

## Kapcsolódó anyagok

- `man ssh`
- *SSH, The Secure Shell: The Definitive Guide* (O'Reilly)
- [#50] Alagutazás: IP-IP tokozás
- [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal
- [#80] Saját felsőszintű tartomány üzemeltetése
- A jelszavak és az SSH-kulcsok kezeléséről http://www.linuxvilag.hu/



# A parancsfájlok

## 72–75. trükk

Előfordulhat, hogy valami olyasmit szeretnénk, amit nemigen lehet egyetlen parancssorban leírni. Ha pedig egyszer-kétszer belebonyolódtunk ilyesmibe, minden bizonnyal elgondolkoztunk már azon nincs-e valamilyen egyszerűbb út (ami megkímélne minket a rengeteg gépeléstől). Általában ez az a pont, amikor a rendszergazdából lassacskán a problémamegoldók különös alfaja kezd kialakulni, amit *programozó* gyűjtőnéven ismerünk.

Ez a fejezet nem gyorstalpaló programozóképző, inkább csak rövid ízelítőt ad a programozás művészetében alkalmazott trükkökből. Úgy vélem, hogy a programozás elsajátításának legjobb módja, ha bátran nekikezdünk. Legegyszerűbben oly módon indulhatunk el, ha megnézzük mások miképpen oldottak meg hasonló feladatokat. A fejezetben szereplő példák önmagukban is hasznosak, de legalább ilyen jól igénybe vehetők saját testreszabott fejlesztéseink megalapozásához. Még a parancsfájlkészítés öreg rókáinak is érdemes egy pillantást vetniük a példákra, ugyanis néhány kevésbé ismert függvényhívásra és nyelvi képességre bukkanhatnak, amelyekkel kevesebb munkával többet végezhetünk.

# #72 Intézzük el gyorsan feladatainkat a movein.sh segítségével

## Hangoljuk össze valamennyi gépünket a helyi környezeti beállításokkal

Ha tartósan használunk egy gépet, idővel elkerülhetetlenül a saját szájízünkre igazítjuk. Amint azt a [#10] „Otthonos parancssori környezet" című trükkben láthattuk, a héjprogram környezete hihetetlenül rugalmas eszköz, amit saját igényeinknek megfelelően alakíthatunk ki.

Általában ezek az apróságok egyszerre több fájlt is érintenek és heteket (vagy akár éveket) vesznek igénybe, mire tökéletesre csiszoljuk őket. Különböző környezeti változókat állítunk be (attól függően, hogy bejelentkező héjprogramról vagy alhéjprogramról van-e szó éppen): szövegszerkesztő beállításokat, levelezési beállításokat, MySQL-értékeket, megfelelő álneveket és egyéb jellemzőket határozunk meg. Amennyiben a környezetet pontosan az ízlésünknek megfelelőre programoztuk, a feladatokat sokkal könnyebb lesz végrehajtanunk és a rendszerrel is élvezetesebb dolgozni.

Amint egy másik gépre lépünk, sajnos mindez egy csapásra semmivé lesz. Iszonyú kiábrándító látni, hogy kedvenc parancsunk kiadása után a `bash: xyz: command not found.` üzenet jelenik meg a képernyőn. Esetleg egy `ls`-t lefuttatva, semmilyen színt nem látunk. De példaként bármely apró kis finomítást felhozhatnánk, ami oly élvezetessé teszi a munkát az otthoni gépen, és amely nélkül a távoli gépen dolgozva kopár, érdektelen környezet vár minket.

Természetesen kézzel átmásolhatjuk beállításainkat, hiszen a legtöbbjük „pontfájlokban" tárolódik a saját könyvtárunkban (mint a *.bashrc* vagy a *.vimrc*), csakhogy elég nehéz az összeset észben tartanunk, ráadásul a klasszikus változatkérdésbe ütközünk: amikor a helyi állományt megváltoztatjuk, az összes távoli állományt is frissíteni kellene. Egyszerűsítsük életünket egy nagyon egyszerű kis parancsfájllal.

### Lista: movein.sh

```
#!/bin/sh

if [ -z "$1" ]; then
echo "Usage: `basename $0` hostname"
```

```
exit
fi

cd ~/.skel
tar zhcf - . | ssh $1 "tar zpvxf -"
```

Programunkat nevezzük *movein.sh-nak*, és helyezzük a *~/bin* könyvtárunkba. Hozzuk létre a *.skel* könyvtárat, és készítsünk közvetett hivatkozást a távoli gépre másolandó fájlokra:

```
rob@caligula:~/.skel$ ls -al
total 12
drwxr-xr-x 6 rob staff 204 Sep 9 20:52 .
drwxr-xr-x 37 rob staff 1258 Sep 9 20:57 ..
lrwxr-xr-x 1 rob staff 11 Sep 9 20:52
➥ .bash_login -> ../.bash_login
lrwxr-xr-x 1 rob staff 11 Sep 9 20:52
➥ .bashrc -> ../.bashrc
lrwxr-xr-x 1 rob staff 11 Sep 9 20:52
➥ .my.cnf -> ../.my.cnf
lrwxr-xr-x 1 rob staff 11 Sep 9 20:52
➥ .pinerc -> ../.pinerc
drwxr-xr-x 3 rob staff 102 Sep 9 20:51 .ssh
lrwxr-xr-x 1 rob staff 9 Sep 9 20:52
➥ .vimrc -> ../.vimrc
lrwxr-xr-x 1 rob staff 6 Sep 9 21:27 bin -> ../bin
```

Figyeljük meg, hogy a *~/.skel/.ssh* kilóg a sorból, ugyanis nem közvetett hivatkozás, hanem valódi könyvtár. *Semmiképpen ne fűzzük a ~/.skel alá a ~/.ssh-t!* A legkevésbé sem hiányzik, hogy az SSH saját kulcsunkat mindenhová szertemásoljuk; pontosan erre tartjuk az *SSH-Agent*-et (lásd [#68] SSH-Agent, hatékonyan). Ehelyett inkább hozzunk létre egy könyvtárat *~/.skel/.ssh* néven, majd a következő közvetett hivatkozásokat készítsük el:

```
rob@caligula:~/.skel$ cd .ssh
rob@caligula:~/.skel/.ssh$ ls -al
total 4
drwxr-xr-x 3 rob staff 102 Sep 9 20:51 .
drwxr-xr-x 6 rob staff 204 Sep 9 20:52 ..
lrwxr-xr-x 1 rob staff 26 Sep 9 20:51
➥ authorized_keys2 ->
  ../../.ssh/id_dsa.pub
```

A hivatkozás neve *authorized_keys2* lesz, és az élő nyilvános kulcsunkra mutat. Ugye nyilvános kulcsokat használunk az SSH-bejelentkezésekhez? Ha nem, feltétlenül olvassuk el a [#66] „Gyors bejelentkezés SSH-ügyfélkulcsokkal" című trükköt.

Mostantól valahányszor lefuttatjuk ezt a parancsfájlt, a *~/.skel* tartalmát a parancssorban megadott gépre másolhatjuk, egyenesen annak saját könyvtárába. A `tar h` kapcsolójának jelentése: „másold ezeket a közvetett hivatkozásokat olyan módon, mintha fájlok és nem hivatkozások lennének", így a távoli végen a hivatkozások tartalmának másolatával fogunk találkozni. Ha esetleg módosítjuk a helyi változatot, csak le kell futtatnunk a parancsfájlt, amely mindent szépen felülír a távoli gépen.

A fent említett *.ssh* könyvtárat csak akkor kell felhasználnunk, ha önműködően, jelszó nélkül szeretnénk belépni a távoli gépre. Mindaddig, amíg a helyi gépünk kulcsai biztonságban vannak, nem vállalunk kockázatot azáltal, hogy további *authorized_keys2* fájlokat készítünk, hiszen éppen erről szól a nyílt kódú titkosítás.

### Kapcsolódó anyagok

- [#66] Gyors bejelentkezés SSH-ügyfélkulcsokkal
- [#68] SSH-Agent, hatékonyan
- [#10] Otthonos parancssori környezet
- SSH: The Secure Shell, The Definitve Guide (O'Reilly)

## #73 Keresés és helyettesítés Perllel

### Dolgozzunk tetszőleges Perl-helyettesítésekkel fájlokon és folyamokon parancsfájlok nélkül

A Perlt néhány kapcsoló segítségével igen hasznos parancssori szerkesztőeszközzé varázsolhatjuk. Ismerkedjünk meg ezekkel a kapcsolókkal, dörmögjünk mágikus Perl-egysorosokat, és máris kiválthatjuk ismerőseink csodálatát (és riogathatjuk projektvezetőnket).

Az első ilyen kapcsoló a -e. Adjuk meg a Perlnek a -e kapcsolót, majd egy sor kódot, és úgy fogja lefuttatni, mintha szabályos Perl-parancsfájl lenne:

```
rob@catlin:~$ perl -e 'print "Szia, mama!\ n"'
Szia, mama!
```

Figyeljük meg, hogy az egysorosoknál nem szükséges kitennünk a sorvégzáró ; (pontosvesszőt). Általában nem árt, ha az egysorost egyszeres idézőjelek (aposztrófok) közé helyezzük, mert így elkerülhetjük, hogy a parancsértelmező feldolgozza a különleges karaktereket (mint a fenti példában a ! (felkiáltójel) és a \ (perjel)).

A következő kapcsoló kicsit összetettebb, de szinte ösztönünkké lesz, ha megszoktuk. Ez a -p kapcsoló. A Perl leírás szerint:

*„A* -p *hatására a Perl az alábbi ciklust feltételezi a programunk körül, így az a* sed*-hez hasonlóan végiglépked valamennyi fájlnév-jellemzőn."*

```
LINE:
while (<>) {
... # your program goes here
}  continue {
print or die "-p destination: $!\ n";
}
```

A kérdéses sor ($_) önműködően kiíródik minden egyes ciklusban. Ha a -e és -p kapcsolókat egyszerre használjuk, olyan egysorost kapunk, amely a parancssor minden fájlján vagy az STDIN csatornán keresztül fogadott szöveg minden során végiglépked. Példaképpen álljon itt egy rettentően bonyolult cat parancs:

```
rob@catlin:~$ perl -pe 1 /etc/hosts
#
# hosts This file describes a number
# of hostname-to-address
# mappings for the TCP/IP subsystem.
#

# For loopbacking.
127.0.0.1 localhost
```

Az `1` itt nem más, mint a visszatérési érték (mintha a Perl parancsfájlban az `1;` utasítást adtuk volna ki, amely egyenértékű a `return 1;` sorral). Mivel a sorok önműködően kiíródnak valójában nincs is szükségünk a `-e` kapcsolóval megadott programra.

A dolog akkor lesz érdekes, amikor az utasítást megfűszerezzük egy kis kóddal, a kiíratás előtt módosítva a sorokat. Tegyük fel, hogy a *localhost* sorhoz szeretnénk hozzáfűzni a helyi gép nevét:

```
rob@catlin:~$ perl -pe 's/localhost/localhost
➥ $ENV{ HOSTNAME} /' /etc/hosts
#
# hosts This file describes a number
# of hostname-to-address
# mappings for the TCP/IP subsystem.
#

# For loopbacking.
127.0.0.1 localhost catlin.nocat.net
```

Esetleg `inetd`-beállításainkat akarjuk módosítani:

```
rob@caligula:~$ perl
➥ -pe 's/^(\ s+)?(telnet|shell|login|exec)/# $2/' \
  /etc/inetd.conf
```

A fenti sor a */etc/inetd.conf* tartalmát írja ki a `STDOUT`-ra, megjegyzésbe téve valamennyi *telnet*, *shell*, *login* vagy *exec* sort, amely még nincs eleve megjegyzésben. Természetesen mindezt visszairányíthatnánk egy fájlba, de ha helyben szeretnénk szerkeszteni a fájlokat, jobb megoldás is létezik: a `-i` kapcsoló.

A `-i` kapcsoló lehetővé teszi, hogy a fájlokat helyben szerkesszük át. Ha tehát a fenti sorokat megjegyzésbe szeretnénk tenni a */etc/inetd.conf* állományban, a következőt próbáljuk ki:

```
root@catlin:~# perl -pi
➥ -e 's/^(\ s+)?(telnet|shell|login|exec)/# $2/'
➥ /etc/inetd.conf
```

vagy még inkább:

```
root@catlin:~# perl -pi.orig
➥ -e 's/^(\ s+)?(telnet|shell|login|exec)/# $2/'
➥ /etc/inetd.conf
```

A második példában */etc/inetd.conf.orig* néven mentjük az eredeti */etc/inetd.conf* állományt, és csak ezután módosítjuk. Ugyanilyen könnyen szerkeszthetünk egyszerre több állományt is. A parancssorban ugyanis több fájlnevet (vagy helyettesítő (joker) karaktert) adhatunk meg:

```
rob@catlin:~$ perl -pi.bak
➥ -e 's/bgcolor=#ffffff/bgcolor=#000000/i' *.html
```

A fenti sor a pillanatnyi könyvtárban található valamennyi HTML-oldalunk háttérszínét fehérről feketére változtatja. Ne feledjük le a sor végén álló *i* betűt, hiszen ez teszi a keresést kis- és nagybetű függetlenné (így a bgcolor="#FFFFFF", de akár a BGColor="#FfFffF" mintát is megtalálja).

Mit tegyünk, ha éppen egy CVS projekt fejlesztésének közepén vagyunk és meg akarjuk változtatni az adatokat fogadó CVS-kiszolgálót? A megoldás egyszerű, továbbítsuk a `find` kimenetét a `perl -pi -e` parancsot futtató *xargs*-on:

```
schuyler@ganesh:~/nocat$ find -name Root |
➥ xargs perl -pi
➥ -e 's/cvs.regikiszolgalo.hu/cvs.ujkiszolgalo.hu/g'
```

Következő lépésként állítsuk be újra a $CVSROOT értékét, majd a megszokott módon jelentkezzünk be a CVS-be, és látni fogjuk, hogy a projektünk csodálatos módon a *cvs.ujkiszolgalo.hu*-ra jelentkezett be.

A Perl parancssros használatával rengeteg hasznos és hatékony módosítást vihetünk véghez röptében. Tanulmányozzuk a szabályos kifejezéseket, használjuk őket bölcsen, és rengeteg kézi szerkesztéstől kímélhetjük meg magunkat.

### Kapcsolódó anyagok

- A Perl programozása (O'Reilly)
- *perldoc perlrun*
- [#10] Otthonos parancssori környezet
- [#30] CVS-modulok megváltoztatása

## #74 Szeleteljük adatainkat kezelhető darabokra (bash)

### Vágjuk kezelhető darabokra nagy méretű bináris állományainkat a bash matematikai képességeivel és a dd segítségével

Lássunk egy példát arra, hogyan vághatunk szét bármilyen állományt tetszőleges darabokra környezeti változók és a bash matematikai értelmezőjének segítségével.

### Lista: mince

```
#!/bin/bash

if [ -z "$2" -o ! -r "$1" ]; then
echo "Usage: mince [file] [chunk size]"
exit 255
fi

SIZE=`ls -l $1 | awk '{ print $5} '`

if [ $2 -gt $SIZE ]; then
echo "Your chunk size must be smaller
      ➥ than the file size!"
exit 254
fi

CHUNK=$2
TOTAL=0
PASS=0
while [ $TOTAL -lt $SIZE ]; do
PASS=$((PASS + 1))
echo "Creating $1.$PASS..."
dd conv=noerror if=$1 of=$1.$PASS
➥ bs=$CHUNK skip=$((PASS - 1)) count=1 2> /dev/null
```

```
TOTAL=$((TOTAL + CHUNK))
done

echo "Created $PASS chunks out of $1."
```

Figyeljük meg, hogy itt kiaknáztuk a `conv=noerror` lehetőséget, hiszen az utolsó darab majdnem teljesen bizonyosan a fájl vége előtt befejeződik. A kapcsoló hatására a `dd` vidáman folytatja a bitek írását, amíg ki nem fogyunk a forrásfájlból, akkor viszont kilép (de nem ad hibaüzenetet, és ami még fontosabb, nem tagadja meg az utolsó darabka kiírását).

Jól jön az ilyesmi, ha például nagy fájlokat akarunk hajlékonylemez (ziplemez, CD-R, DVD, Usenet) méretű darabokra szeletelni a mentés előtt. Mivel a `dd skipp` képességét használjuk ki, tetszőleges méretű fájlokat tudunk kezelni a rendelkezésre álló memóriától függetlenül (feltételezve, hogy értelmes darabkaméretet adtunk meg). Mivel a blokkméretet (*bs*) éppen akkorára állítjuk, mint amekkora darabkaméretet választottunk, viszonylag gyorsan fog futni, különösen, ha a darabkák néhány kilobájtnál nagyobbak.

A darabkákat a pillanatnyi könyvtárba, külön állományokba menti (a név végéhez fűzve a . (pontot) és a darabka sorszámát). Az `ls` *FÁJLNÉV*`.*` parancs segítségével számsorrendben tekinthetjük meg a darabkákat. De mi a helyzet, ha kilencnél több darabkánk keletkezik?

```
ls FÁJLNÉV.* | sort -n -t . +2
```

Hogyan tudjuk összeállítani őket?

```
cat `ls FÁJLNÉV.* | sort -n -t . +2`
➥ > FÁJLNÉVs.complete
```

(Még elegánsabb megoldás, ha darabkák sorszámát mindig négy számjegyre formázzuk. Ilyenkor ugyanis sokkal egyszerűbben, a `cat` `ls` *FÁJLNÉV*`.*` `>` *FÁJLNÉV*`.kész` paranccsal összeállíthatjuk az eredeti állományt. Ehhez a szeletelő parancsfájl magját a következőképpen kell módosítani:

```
dd conv=noerror id=$1 of=$1.`printf "%04i" $PASS`
➥ bs=$CHUNK skip=$((PASS - 1)) count=1 2> /dev/null
```

*– A fordító.)*

Bizonyítékot szeretnél?

```
diff FÁJLNÉV FÁJLNÉV.kész
```

(Természetesen mindez csak akkor működik helyesen, ha fájlneveinkben csak egyetlen pont található. Amennyiben *rettentően.hosszú.és.sok.pontot.tartalmazó* fájlneveket akarunk használni, nézzük meg a *short(1)* súgóoldalát.)

## #75 Színes naplóelemzés terminálablakban

### Jelenítsük meg naplóinkat pompás színekben, xterm ablakban

Egyszer csak elérkezik az a pillanat, amikor azt érezzük, hogy mindjárt kancsalságot kapunk a rendszernaplófájlok böngészésétől, ekkor ideje valamilyen segédeszközt munkába állítanunk, ami segítségünkre siet naplóink rendszerezésében. Egy helyesen beállított rendszernapló (ahogy azt a [#54] „Hogyan lehet a syslogot keményebb munkára sarkallni, és miképpen lehet kevesebb időráfordítással hatalmas naplóállományokat áttekinteni?" című trükkben láthattuk) sokat lendíthet a naplófájl érthetőségén, de még így is elég kemény feladat több megabájtos */var/log/messages* fájlokban kotorászni bizonyos minták után.

A színes `ls` első ránézésre segít megkülönböztetni a fájltípusokat. Nem lenne rossz, ha lenne egy színes `grep` is, amellyel kiemelhetnénk egy-egy mintát a szürke sorok tengeréből. Számtalan X-alkalmazás létezik, amely pontosan ezt teszi, de miért ne oldjuk meg mindezt parancssorból?

A következő kódot *~/bin/rcg* néven mentsük (rcg, azaz a Regex Colored Glasses, vagyis a szkiffel színezett szemüveg rövidítése):

```
#!/usr/bin/perl -w
use strict;
use Term::ANSIColor qw(:constants);
```

```
my %target = ( );

while (my $arg = shift) {
my $clr = shift;

if(($arg =~ /^-/) | (!$clr)) {
print "Usage: rcg [regex] [color] [regex]
➥ [color] ...\ n";
exit;
}

#
# Ugly, lazy, pathetic hack here
#
$target{ $arg}  = eval($clr);
}

my $rst = RESET;

while(<>) {
foreach my $x (keys(%target)) {
s/($x)/$target{ $x} $1$rst/g;
}
print;
}
```

Az *rcg* tulajdonképpen egy egyszerű szűrő, ami a parancssorban megadott tetszőleges szkifek színezésére a *Term:ANSIColor*-t használja fel. Célja, hogy a naplófájlokat szemrevételezve könnyebben értelmezhessük a sorokat.

Az *rcg* mindig páros számú parancssori kapcsolót vár. A páratlan kapcsolók határozzák meg a szkifeket, a párosak pedig a színeket adják meg.

Tegyük fel, hogy minden sort, amely a *Sendmail* szót tartalmazza lilával szeretnénk kiemelni a szürke tömegből:

```
$ rcg sendmail MAGENTA < /var/log/messages |
➥ less -r
```

A `less -r` elhagyható, de hasznos (mivel így `less` alatt a kívánt színek jelennek meg a hozzájuk tartozó ESC karakterek helyett.)

Páros kapcsolóként tetszőleges szkifet használhatunk:

```
$ rcg '\ d+\ .\ d+\ .\ d+\ .\ d+' GREEN
➥ < /var/log/maillog
```

vagy akár össze is fűzhetjük a színeket:

```
$ tail -50 /var/log/messages |
➥ rcg WARNING 'BOLD . YELLOW . ON_RED'
```

Egyetlen parancssorban tetszőleges számú szkif, illetve szín párost adhatunk meg. Ez az a hely, ahol az apró parancsfájlok vagy álnevek roppant hasznosnak bizonyulhatnak (egy az üzenetekhez, egy a tűzfalnaplókhoz, egy az Apache-hoz).

A színek és kombinációk teljes listáját a *Term:ANSIColor* leírásában találjuk.

Néhány hasznos karaktersorozat:

`\w+=\S+`
: változók, mint például a `TERM=xyz` kifejezés,

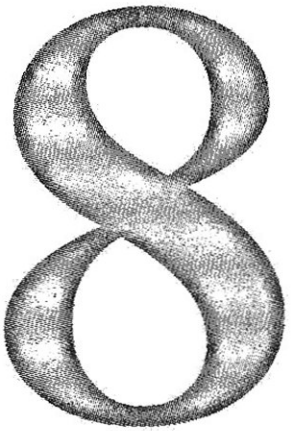
`\d+\.\d+\.\d+\.\d+`
: valószínűleg IP-cím,

`^(J|F|M|A|S|O|N|D)\w\w (\d|)\d`
: könnyen lehet dátum,

`\b\d\d:\d\d:\d\d\b`
: valószínűleg az idő,

`.*last message repeated.*`
: ezt állítsuk `BOLD . WHITE` értékre.

Dolgoztassuk meg képzeletünket, de ne feledjük, hogy a színkapcsolókat `eval()`-lal értelmeztük. Elméletileg rengeteg érvényes Perl-kifejezést helyettesíthetünk be a szkifek vagy a színek helyére; ennek a hmm, *képességnek* a kiaknázását az olvasóra bízzuk. Mindenesetre jobban tesszük, ha nem futtatunk le mindenféle *rcg* sorokat rendszergazdai jogosultsággal, hacsak nem magunk gépeltük be őket. Továbbá figyeljük meg, hogy a színezés tetszőleges sorrendben végezhető, így az átfedő szkifek viselkedését lehetetlen előre megmondani.



# Adatbázis-kiszolgálók

## 76–100. trükk

Adatkiszolgálók építéséhez kitűnő választás a Linux. Maguk az adatkiszolgálók azonban ritkán részei a Linuxnak. Általában a Linux egyszerűen csak „létfenntartó készülékként" működik az adott célnak szentelt összetettebb adatszolgáltatások mögött.

Az utolsó fejezetben három nagyobb alkalmazást vizsgálunk meg közelebbről. Mindhárom nagyon jól fut Linux alatt, és ma már az internet adatszolgáltatásainak gerinceként tarthatjuk őket számon. A BIND (amelyet Internet Software Konzorcium fejlesztett ki) messze a legelterjedtebb DNS adatkiszolgáló az egész bolygón. Feladata az internetet működtető tartománynevek és IP-címek közötti fordítás. Ha általánosabb adatforrást keresünk, vessünk egy pillantást a MySQL-re, erre a pehelysúlyú, gyors és méretezhető SQL adatbázis-kiszolgálóra, amely számos hálózati kereskedelmi, üzleti alkalmazás alapjául szolgál. S amikor azután a felhasználóknak át kell adni az adatot, belép a képbe az Apache, ami messze a legelterjedtebb webkiszolgáló a földkerekségen. Több kiszolgálón fut Apache, mint az összes többi ismert webkiszolgáló gépen *együttvéve,* és nem ok nélkül: érett, üzembiztos, gyors, valamint hasznos és érdekes szolgáltatásokkal telezsúfolt kiszolgálóról van szó.

Amennyiben a csomagok futtatásával kapcsolatban további útmutatásra van szükségünk, olvassuk el a hozzájuk tartozó hálózati dokumentációkat. A jelenlegi internet jó része ezek alatt az alkalmazások alatt fut, így széles körű és részletes leírást találunk róluk. Ezenkívül a O'Reilly kiadó mindhárom alkalmazásról adott már ki kitűnő könyveket: *DNS and Bind, MySQL Reference Manual* és *Apache: The Definitive Guide* címmel, hogy néhányat említsünk. Paul DuBois *MySQL* című írása (New Riders) szintén kiváló bemutató a MySQL futtatásához.

A következő trükkökben olyan, nem feltétlenül egyértelmű módszereket fogunk látni, amelyek segítségével kiszolgálóink úri kedvünk szerint továbbítják majd az adatokat. Megnézünk néhány módszert, amellyel nagy méretű telepítéseket is kezelhetünk, miközben a lehető legkevesebb időt fordítjuk még az igen bonyolult rendszerek karbantartási munkáira is.

## #76 BIND futtatása chroot gyökércserével

### A chroot megfontolt használatával tartsuk elzárva named démonunkat a rendszer többi részétől

Az internet nagy részének névfeloldási rendszere a BIND programtól függ. Bár hatalmas erőfeszítésekkel próbálták a BIND valamennyi lehetséges biztonsági hibáját kiküszöbölni, soha nem lehetünk teljesen bizonyosak benne, hogy egy kód tökéletesen biztonságos. Hogy a lehető legkisebbre csökkenthessék az esetleges (veremtúlcsordulás, hiba vagy rossz beállítások miatt bekövetkező) rendszerfeltörés által okozott kárt, a BIND kiszolgáló ma már `named` felhasználóként fut, és nagyszámú gépen módosított gyökérkönyvtárral (`chroot`) futtatják a névkiszolgálót. Bár a gyökércsere nem feltörhetetlen, azonban megfelelő felépítése esetén igen nehéz dolga lesz a gonosz támadóknak.

A következőkben bemutatjuk, hogy milyen lépéseket kell megtenni, amennyiben mi is `chroot` gyökércserével szeretnénk futtatni a BIND 9-et. A DNS biztonság hatalmas és összetett témakör, amit nem szabad félvállról venni. További tájékoztatást a fejezet végén felsorolt források közt találunk.

Először is el kell érnünk, hogy a named ne rendszergazdai jogosultsággal, hanem valamilyen más felhasználónév alatt fusson. Készítsünk külön felhasználót és csoportot, amely alatt majd a named folyamatot működtetjük:

```
root@gemini:~# groupadd -g 53 named
root@gemini:~# useradd -u 53 -g named -c "chroot
➥ BIND user" \
  -d /var/named/jail -m named
```

Induláskor a named démont a */var/named/jail* könyvtárba utasítjuk. Ez alatt a könyvtár alatt el kell készítenünk azt a könyvtár-vázszerkezetet, amely elegendő a named szabályos indulásához. Készítsük el a */var* szerkezetet és másoljuk ide DNS-adatainkat.

```
root@gemini:~# cd ~named
root@gemini:/var/named/jail# mkdir -p var/{
➥ run,named}
root@gemini:/var/named/jail# cp -Rav
➥ /var/named/data var/named/
```

Ha bármely zónában másodlagos (slave) módban működünk, akkor a named folyamatnak meg kell adnunk az írási jogosultságot is a könyvtárra (hogy a másodlagosként kapott frissített adatokat lejegyezhesse). Készítsünk külön könyvtárat kifejezetten a másodlagos adatoknak, szabályos adatfájljainktól függetlenül:

```
root@gemini:/var/named/jail# mkdir var/named/slave
root@gemini:/var/named/jail# chown named.named
➥ var/named/slave
```

Következő lépésként készítsük el a *dev/* és *etc/* könyvtárakat, majd másoljuk ide a létfontosságú fájlokat és eszközöket:

```
root@gemini:/var/named/jail# mkdir { dev,etc}
root@gemini:/var/named/jail# cp
➥ -av/dev/{null,random}dev/
root@gemini:/var/named/jail# cp -av \
  /etc/{ localtime,named.conf,rndc.key}  etc/
```

Töröljük a fájlok jogosultságait:

```
root@gemini:/var/named/jail# chown root.root .
root@gemini:/var/named/jail# chmod 0755 .
root@gemini:/var/named/jail#
➥ chown named.named var/named/data/
root@gemini:/var/named/jail#
➥ chmod 0700 var/named/data/
root@gemini:/var/named/jail#
➥ chown named.named var/run/
```

Ha a DNS naplókhoz *syslog*-ot használunk, a *syslog* indító *rc* fájlját ki kell egészítenünk egy kapcsolóval. Ezáltal tudatjuk a *syslog*-gal, hogy a megadott foglalatot (socket) is figyelje (ezt muszáj megadnunk, hiszen a rendszer */dev/log* állománya a gyökércseréből nem érhető el):

```
syslogd -m 0 -a /var/named/jail/dev/log
```

Ha fájl rendszernaplókat használunk, nem kell megváltoztatnunk a *syslogd* beállításait, csak azt ellenőrizzük, hogy a fájlokat írhatja-e a named felhasználó, és létezik-e *log* könyvtár a */var/named/jail/* alatt.

Végül indítsuk el a named démont a parancssorban, átadva a felhasználónevet, a *chroot* könyvtárat és az indító beállításfájlt:

```
root@gemini:~# /usr/sbin/named -u named -t
/var/named/jail \
  -c /etc/named.conf
```

Amennyiben a named nem indulna zökkenőmentesen, nézzük át a */var/log/syslog* vagy a */var/log/messages* naplókat, és derítsük ki a hiba okát. Nem árt, ha meggyőződünk róla, hogy minden a megfelelő jogosultsággal rendelkezik-e. A következőben egy *chrootolt* BIND-ot futtató gép rekurzív ls kimenetét láthatjuk:

```
root@gemini:/var/named/jail# ls -lR
.:
total 12
drwxr-xr-x 2 root root 4096 Sep 20 12:45 dev/
drwxr-xr-x 2 root root 4096 Sep 20 13:08 etc/
drwxr-xr-x 5 root root 4096 Sep 20 13:04 var/
```

```
./dev:
total 0
crw-rw-rw- 1 root root 1, 3 Jul 17 1994 null
crw-r--r-- 1 root root 1, 8 Dec 11 1995 random

./etc:
total 16
-rw-r--r-- 1 root root 1017 Sep 20 12:46 localtime
-r--r--r-- 1 root root 1381 Sep 20 13:08 named.conf
-rw------- 1 root root 77 Sep 11 04:22 rndc.key

./var:
total 12
drwxr-xr-x 4 root root 4096 Sep 20 13:01 named/
drwxr-xr-x 2 named named 4096 Sep 20 13:15 run/

./var/named:
total 8
drwx------ 3 named named 4096 Sep 20 13:03 data/
drwxr-xr-x 2 named named 4096 Sep 20 12:43 slave/

./var/named/data:
total 42
-rw-r--r-- 1 root root 381 Apr 30 16:32
localhost.rev
-rw-r--r-- 1 root root 2769 Sep 20 13:03 named.ca
-r--r--r-- 1 root root 1412 Sep 17 16:44 nocat.net

./var/named/slave:
total 0

./var/run:
total 4

-rw-r--r-- 1 named named 6 Sep 20 13:15 named.pid
```

A *chroot*-olt környezetek lehetővé teszik, hogy démonjainkat a rendszer többi részétől elzárva futtassuk, de azért ne higgyük, hogy ezzel egy csapásra elértük a tökéletes biztonságot. Ez a módosítás csak az első lépés, mindenképpen érdemes elolvasni a BIND futtatásáról és használatáról szóló írásokat.

### Kapcsolódó anyagok

- `http://www.losurs.org/docs/howto/Chroot-BIND.html`
- DNS Bind, Fourth edition (O'Reilly)

## #77 Nézetek BIND 9 alatt

### A különböző helyekről érkező kérésekre adjunk különböző válaszokat

Egészen mostanáig, ha az egyik gépcsoportnak más adatokat akartunk nyújtani, mint a másiknak, akkor több DNS-kiszolgálót kellett futtatnunk, esetleg trükkös beállításokkal, egyetlen gépen több névkiszolgáló-folyamatot indítottunk el. Senki se mondta, hogy könnyű dolog kétarcúnak lenni!

A BIND 9 azonban bevezetett egy új, nézeteknek (views) nevezett lehetőséget, amellyel a különböző zónaváltozatok vagy akár a névkiszolgáló beállítások kezelése jelentősen leegyszerűsödik.

### Alapvető írásmód

A nézetek beállításának alapja a BIND 9-ben megjelent *view* (nézet) kulcsszó. A *view* egy nézetnevet vár értékként és egyetlen kötelező alkulcsszava van: mégpedig a *match-clients*. A nézet neve egyszerűen csak a nézet azonosítására szolgáló kényelmes rövidítés; én általában olyasmiket használok, mint az *internal*, *external* vagy *internet*. Ha olyan szót szeretnénk használni, amely ütközik a *named.conf* valamelyik foglalt szavával, ne felejtsük el idézőjelek közé tenni. Mivel én sohasem emlékszem melyik szó védett és melyik nem, biztos, ami biztos alapon, mindet idézőjelbe szoktam tenni.

A *match-clients* alkulcsszó-értékként címlistát vár. Csak azok a lekérdezések fognak hozzáférni a nézetben található beállításokhoz, amelyek IP-címe összeegyeztethető a megadott címlistával. Amennyiben a kérdező IP-címe több nézet *match-clients* listájával is egyezik, az első találatot elérő nézetkulcsszó lesz érvényes.

Lássunk egy egyszerű példát:

```
view "global" {
match-clients {  any; } ;
} ;
```

Ez a nézet semmi hasznosat nem művel, mivel az összes kérelemre válaszol (más *view* kulcsszavak híján) és a `match-clients` utasításon kívül egyetlen alkulcsszót sem tartalmaz, amely megváltoztatná az alapértelmezett beállításokat.

Amennyiben egy nézetben nem adunk meg semmilyen alkulcsszót, a nézet az alnézetre vonatkozó globális beállításokat örökli. Így például, ha valamelyik nézetben nem kapcsoljuk ki a rekurziót, akkor az örökölni fogja az `options` beállításunk rekurzió beállításait (amelyet a *recursion* alkulcsszó alatt találunk), illetve ha ilyenünk sincsen, örökli a globális alapértelmezett értéket, amely a rekurzió esetében *yes* (azaz beállított) lesz.

Lássunk egy példát a rekurzió kikapcsolására:

```
view "external" {
match-clients {  any; } ;
recursion no;
} ;
```

A következőkben bemutatunk egy teljes *named.conf* állományt, amely tartalmazza a fenti nézetmegadást is:

```
/*
* BIND 9 name server allowing recursive queries
from
* localhost, disallowing from anywhere else
*/

options {
directory "/var/named";
} ;
```

```
view "localhost" {
match-clients {  localhost; } ;
recursion yes; /* this is the default */
} ;

view "external" {
match-clients {  any; } ;
recursion no;
} ;
```

Ez a névkiszolgáló csak a saját gépünkről érkező rekurzív lekéréseket engedélyezi. (A *localhost* hozzáférés-vezérlés lista (Access Control List, azaz ACL) értéke előre meghatározott: a gép összes névkiszolgálót futtató IP-címe, továbbá a *loopback* (hurokeszköz) cím tartozik bele.) Semmilyen más címről érkező kérést nem tekintünk rekurzívnak.

Nézzünk egy hasonló beállítást, ahol a névkiszolgáló a belső hálózatunkon engedélyezi a rekurzív lekérést, az internetről viszont letiltja:

```
/*
* Same name server serving an internal network
* and the Internet
*/

options {
directory "/var/named";
} ;

view "internal" {
match-clients {  localnets; } ;
recursion yes; /* this is the default */
} ;

view "external" {
match-clients {  any; } ;
recursion no;
} ;
```

Ez a kiépítés a szintén beépített *localnets* ACL-t használja ki, amely az előmegatározás értelmében azokat a hálózatokat jelenti, amelyekhez a névkiszolgáló közvetlenül csatlakozik.

### Zónák megadása a nézetekben

Ha azt szeretnénk, hogy egy zónaváltozatot csak néhány lekérdező láthasson, akkor azt külön nézetben kell megadnunk. A zónakulcsszót nagyon egyszerűen a nézet alkulcsszavaként adhatjuk meg. Minden egyéb tekintetben azonos szabályok szerint használjuk, mint felső szintű kulcsszót. Felhívjuk a figyelmet rá, hogy amennyiben a nézeten belül megadunk valamilyen zónát, akkor ugyanabban a nézetben az összes többit is meg kell adnunk.

Lássunk egy példát!

```
/*
* A name server showing different zone data to
* different networks
*/

options {
directory "/var/named";
} ;

view "internal" {
match-clients {  localnets; } ;
recursion yes; /* this is the default */

zone "oreilly.com" {
type master;
file "db.oreilly.com.internal";
allow-transfer {  any; } ;
} ;
} ;

view "external" {
match-clients {  any; } ;
recursion no;

zone "oreilly.com" {
type master;
file "db.oreilly.com.external";
allow-transfer {  none; } ;
} ;
} ;
```

Figyeljük meg, hogy az *oreilly.com* tartományt az external és az internal zónában egyaránt megadtuk, de az internal nézeteknél a *db.oreilly.com.internal* zónaadatfájlt, az external nézeteknél pedig a *db.oreilly.com.external* adatfájlt használjuk. Feltételezhetően a két adatfájl tartalma nem azonos.

Amennyiben szívesebben használnánk különböző alkönyvtárakat az eltérő zónákhoz, természetesen ezt is megtehetjük. Például az internal nézet *oreilly.com*-hoz tartozó fájl alkulcsszava lehetne az *internal/db.oreilly.com;* external párja pedig *external/db.oreilly.com.*

Ilyen módon valamennyi belső zónaadatfájlunkat az *internal* könyvtárban, míg az összes külsőt az *external* könyvtárban tarthatjuk.

### Másodlagos névkiszolgálók nézetei

A nézetek beállításakor némi gondot okozhat a másodlagos névkiszolgálók beállítása. Elég sokan többszörös nézettel készítik el az elsődleges névkiszolgálójukat, és szeretnék, ha a másodlagos kiszolgálók is ugyanezeket a beállításokat tartanák meg. Amikor a másodlagos kiszolgáló megpróbálja letölteni a zónákat (vagy egyazon zóna két változatát) az elsődleges névkiszolgálóról, sajnos csak az egyik zónaváltozatot láthatja, azt, amelyiket a másod IP-címéhez beállított nézetben talál.

E nehézséget úgy oldhatjuk meg, hogy a másodlagos névkiszolgálónak IP-álneveket adunk, így két IP-címe lesz, ezek segítségével már átmásolhatja a különböző zónabeállításokat. Az elsődleges névkiszolgálót állítsuk be olyan módon, hogy a másodlagos kiszolgáló egyik IP-címére az egyik, a másikra a másik zónabeállítást mutassa. Végül rá kell vennünk a másodlagos névkiszolgálót, hogy minden nézetben a megfelelő IP-címről kezdeményezze a letöltést.

Lássunk rá egy példát! A másodlagos névkiszolgálónknak két IP-címe van: 192.168.0.2 és 192.168.0.254. Az elsődleges névkiszolgálónak csak egy, a 192.168.0.1. Először lássuk a másodlagos névkiszolgáló *named.conf* állományát:

```
options {
directory "/var/named";
} ;

view "internal" {
match-clients {  localnets; } ;
recursion yes;s

zone "oreilly.com" {
type slave;
masters {  192.168.0.1; } ;
transfer-source 192.168.0.2;
file "internal/bak.oreilly.com ";
allow-transfer {  any; } ;
} ;
} ;

view "external" {
match-clients {  any; } ;
recursion no;

zone "oreilly.com" {
type slave;
masters {  192.168.0.1; } ;
transfer-source 192.168.0.254;
file "external/bak.oreilly.com ";
allow-transfer {  none; } ;
} ;
} ;
```

Figyeljük meg a másodlagos névkiszolgáló beállítását. Az *oreilly.com* letöltés megkezdése belső nézet esetében a 192.168.0.2 címről történik, míg külső nézet esetén a 192.168.0.254 címről indul a kapcsolat.

Lássuk az elsődleges névkiszolgáló *named.conf* állományát:

```
options {
directory "/var/named";
} ;

view "internal" {
match-clients {  !192.168.0.254; localnets; } ;
recursion yes;
```

```
zone "oreilly.com" {
type master;
file "internal/db.oreilly.com ";
allow-transfer {  any; } ;
} ;
} ;

view "external" {
match-clients {  any; } ;
recursion no;

zone "oreilly.com" {
type master;
file "external/db.oreilly.com ";
allow-transfer {  192.168.0.254; } ;
} ;
} ;
```

Figyeljük meg, hogy az internal nézet *match-clients* alkulcsszava a 192.168.0.254 címet szándékosan áthelyezi a külső listába, hiszen kivonja a belső címek listájából. Valahányszor a másodlagos névkiszolgáló erről az IP-címről kezdeményez zónakapcsolatot, a */var/named/external/db.oreilly.com* zónaadatfájlban leírt *oreilly.com* változatot fogja látni.

## #78 Készítsünk gyorstárazó DNS-t a helyi tartományokhoz!

### Pörgessük fel a BIND-ot: használjunk továbbító és gyorstárazó kiszolgálót

Ha kellően összetett hálózatszerkezettel bírunk, a BIND használata meglehetősen kemény dió lesz. Több DMZ (Demilitarized Zone, azaz szabad zóna) alkalmazása, nyilvános és fenntartott IP-címek, átruházott altartományok, és egy komolyabb helyen máris teljes állású DNS rendszergazdára lesz szükség. Amennyiben valahogy mérsékelni szeretnénk rendszer bonyolultságát, nézzük át a [#77] „Nézetek BIND 9 alatt" részt. Ha igazán kalandvágyók vagyunk, próbálkozzunk meg a helyettesítő karakteres tartománykereséssel és átruházással, amint azt a [#100] „Szuperszolgáltatás: tömeges webhelyszolgáltatás helyettesítő karakterekkel, proxyval és újraírással" trükkben olvashatjuk.

A BIND kis és közepes telepítésekben ezzel szemben tulajdonképpen csak két feladatot lát el: hivatalos forrásként szolgál egy (esetleg két)

tartományhoz, a többi kérést pedig átirányítja más DNS-kiszolgálókhoz. Nézzünk meg egy egyszerű, (de teljes értékű) *named.conf* állományt, ami pontosan ezt a feladatot végzi el:

```
options {
directory "/var/named";
pid-file "/var/run/named.pid";
statistics-file "/var/named/named.stats";
} ;

logging {
channel default_out {
file "/var/log/named.log";
} ;

category default {  default_out; } ;
category config {  default_out; } ;
category xfer-in {  default_out; } ;
category xfer-out {  default_out; } ;
category lame-servers {  null; } ;
} ;

zone "0.0.127.in-addr.arpa" in {
type master;
file "data/localhost.rev";
} ;

zone "." {
type hint;
file "rootservers.cache";
} ;

// Authoritative domains go here

zone "nocat.net" {
type master;
file "data/nocat.net";
} ;
```

A fentiek szerint a *nocat.net* tartományban mi szolgáltatunk neveket a */var/named/data/nocat.net* állományban tárolt adatoknak megfelelően. Ha a *nocat.net* tartományon (vagy a 127.0.0.0 visszacsatolt hálózaton) kívüli kérés érkezik, azt a *rootservers.cache* fájlban megadott

gyökérkiszolgálóknak megfelelően önműködően továbbítjuk. A BIND-dal együtt ugyan érkezik egy jól használható gyökérkiszolgáló-gyorstár, de ha valamiért nem találnánk meg, próbáljuk meg a következőt:

```
# dig @a.root-servers.net > rootservers.cache
```

Amennyiben a hálózatunknak nincs korlátlan internethozzáférése (például mert korlátozó tűzfal mögött vagyunk), a megfelelő kiszolgálóra továbbítás nem fog működni. Ez esetben próbálkozzunk meg közvetlen továbbító szabállyal, az alábbi bejegyzés alapján:

```
zone "." {
type forward;
forward only;
forwarders {  192.168.1.1; } ;
}
```

Természetesen a 192.168.1.1 helyére a hálózatunk érvényes névkiszolgálójának IP-címét kell írnunk. Ez a beállítás az összes DNS-forgalmat a hálózatunk DNS-kiszolgálójára irányítja át, amelynek feltételezhetően van engedélye tartománykeresést végezni a tűzfalon keresztül.

### Kapcsolódó anyagok

- [#77] Nézetek BIND 9 alatt.
- [#100] Szuperszolgáltatás: tömeges webhelyszolgáltatás helyettesítő karakterekkel, proxyval és újraírással.
- DNS & BIND, 4 Edition (O' Reilly)

## #79 Terhelésmegosztás címforgatásos DNS segítségével

### Közvetlen forgalom egyidejűleg több kiszolgálóra, címforgatásos DNS használatával

Amennyiben elegendően népszerű helyet kezelünk, idővel elérkezünk egy ponthoz, amin túl a kiszolgálónk egyszerűen már nem képes eleget tenni a további kéréseknek. A webkiszolgálók világában ezt Slashdot jelenségnek hívják, és nem éppen kellemes dolog. Bár ideiglenesen megoldhatjuk a feladatot erősebb processzor, még több

memória, gyorsabb buszok és lemezek vásárlásával, de idővel esetleg rá kell ébrednünk, hogy egyetlen gép egyszerűen képtelen ellátni a feladatot.

Az egyik megoldás, amellyel úrrá lehetünk a monolitikus kiszolgáló-megoldás okozta hibákon, ha megosztjuk a munkát több gép közt. Hogyha egy második (esetleg harmadik) gépet adunk az elérhető gépek csoportjához, nemcsak a hálózat sebességét, de a megbízhatóságát is jelentősen növelhetjük, hiszen ha van egy (vagy több) folyamatosan üzemelő tartalékunk, az egyik leállásakor a többiek még mindig átvehetik a munkát.

A hálózati terhelés megosztására talán az a legegyszerűbb megoldás, ha magára a nyilvánosságra bízzuk annak kezelését. A címforgatásos (round-robin) DNS mágikus segítségével az egyetlen névre érkező hívásokat több különböző IP-címhez irányíthatjuk át.

BIND 9 alatt ez mindössze annyiból áll, hogy egyetlen gazdagéphez több A bejegyzést veszünk fel. Tételezzük fel, hogy a következő bejegyzéseket használjuk az *oreillynet.com* zónafájljában:

```
www 60 IN A 208.201.239.36
www 60 IN A 208.201.239.37
```

Mostantól kezdve, ha egy gép a *www.oreillynet.com* nevet keresi DNS-kiszolgálónkon, az esetek felében ezt látja majd:

```
rob@caligula:~$ host www.oreillynet.com
www.oreillynet.com has address 208.201.239.36
www.oreillynet.com has address 208.201.239.37
```

Ugyanakkor az esetek másik felében ezt kapja:

```
rob@caligula:~$ host www.oreillynet.com
www.oreillynet.com has address 208.201.239.37
www.oreillynet.com has address 208.201.239.36
```

Mivel a legtöbb alkalmazás csak a DNS által visszaadott első számot használja fel, a módszer szépen működik. Körülbelül a kérelmek fele kerül mindegyik kiszolgálóra, így a két gép közötti terhelés közelítőleg fele-fele arányban oszlik meg. A TTL értéket alacsonyra állítjuk (60 másodpercre), nehogy az útközben található gyorstárazó DNS-kiszolgálók túl sokáig beragadjanak az egyik címnél, így reményeink szerint a két gépre érkező kérelmek számát nagyjából azonos szinten tudjuk tartani.

Mindez csak akkor használható a terhelés megosztására, ha a kiszolgálók egymás közt rendezik, hogy mit is szolgáltatnak. Ha az adataink nem frissülnek, akkor előfordulhat, hogy a böngészők elsőre az egyik kiszolgálóról töltik le az egyik változatot, majd az újratöltés kérésekor egy másik oldalváltozat jön be. Ha olyan módszert keresünk, amivel elkerülhetjük az ilyesmit, olvassuk el a [#41] „Fájlrendszerek egyes részeinek összehangolása az rsync paranccsal" trükköt.

Azt se felejtsük el, hogy az IP-cím átvétele (ha az egyik gép nem képes a feladatait ellátni) nem éppen egyszerű. Ha az egyik kiszolgáló leáll, nem változtathatunk egyszerűen a névkiszolgáló beállításain megvárva, míg a módosítás végigterjed az interneten. Olyan megoldásra van szükségünk, amellyel a kieső kiszolgálót azonnal ki tudjuk váltani. Egy ilyen megoldásra láthatunk példát a [#63] „IP-átvétel kevés ráfordítással" trükkben.

### Kapcsolódó anyagok

- [#41] Fájlrendszerek egyes részeinek összehangolása az `rsync` paranccsal
- [#63] IP-átvétel kevés ráfordítással

## #80 Saját felsőszintű tartomány üzemeltetése

### A navigáció egyszerűsítése érdekében készítsünk saját TLD-t BIND alatt.

Amennyiben saját címzést használó hálózatot felügyelünk biztosan találkozunk már azzal a skizofrén érzéssel, amikor olyan zónafájlokat szeretnénk létrehozni, amelyek a belső és külső IP-címeket egyaránt

helyesen kezelik. A BIND 9 nézeteinek bevezetésével (lásd [#77] Nézetek BIND 9 alatt) szerencsére jelentősen leegyszerűsödött az azonos tartományon belüli különböző IP-csoportok kezelése.

A nézetek segítségével viszonylag egyszerűen úrrá lehetünk a feladaton, de vizsgáljunk meg egy másik lehetőséget is: a saját felsőszintű tartomány felállítását. A *named.conf* zónabejegyzései általában valahogy így néznek ki:

```
zone "oreillynet.com" {
type master;
file "data/oreillynet.com";
} ;
```

A fenti bejegyzés az *oreillynet.com* altartományt kezelő DNS-kiszolgálóhoz tartozik. A jelenlegi legfelső szintű tartományok (.com, .org, .net, .int stb.) kezelését a gyökér DNS-kiszolgálóként ismert titokzatos gépek végzik. Bár így a kiszolgálóink nem lesznek összhangban az internettel, néha mégis hasznos lehet, ha felállítjuk a kizárólag a saját belső hálózatunkra vonatkozó, saját TLD csoportunkat.

Tételezzük fel, hogy a gépeink egy csoportja a 192.168.1.0/24 belső alhálózatot használja. A gépek nincsenek közvetlenül az internetre csatlakoztatva, és DNS-adataikat sem szeretnénk az önjelölt betörők orrára kötni. Próbáljunk felállítani egy nem szabályos TLD-t:

```
zone "bp" {
type master;
file "data/bp";
allow-transfer {  192.168.1/24; } ;
allow-query {  192.168.1/24; } ;
} ;
```

(A bp a BackPlane rövidítése, ebben a példában ezt fogjuk használni.)

Miután a fenti sorokat beírtuk a zónafájlunkba, készítsük el a bp-hez tartozó elsődleges bejegyzést, ahogy azt bármely más tartomány esetében is tennénk:

```
$TTL 86400
@ IN SOA ns.bp. root.homer.bp. (
2002090100 ; Serial
10800 ; Refresh after 3 hours
3600 ; Retry after 1 hour
604800 ; Expire (1 week)
60 ; Negative expiry time
)

IN NS ns.bp.

ns IN A 192.168.1.1

homer IN A 192.168.1.10
bart IN A 192.168.1.11
lisa IN A 192.168.1.12
```

Töltsük újra a named-et, és máris pingelhetjük az új *homer.bp* gépet. Ha azt szeretnénk, hogy más névkiszolgálók is tárolják TLD táblánk másodlagos másolatait, vegyük fel őket a szokásos módon:

```
zone "bp" {
type slave;
file "db.bp";
masters {  192.168.1.1; } ;
} ;
```

Ezzel a módszerrel teljes belső hálózatunkon szétküldhetjük az új TLD-t. Amennyiben interneten keresztül csővezetékeket használunk (ahogy azt az [#50] „Alagutazás: IP-IP tokozás" trükkben olvashattuk), és ilyen módon kapcsoljuk össze a távoli irodákat vagy barátainkat, saját TLD-szerkezetünk elméletileg tetszőleges méretűre növelhető.

## #81 MySQL felügyelete az mtop segítségével

### Jelenítsünk meg MySQL-szálakat topszerű formátumban

A *top* nevű testvéréhez hasonló módon, az *mtop* a MySQL-kiszolgáló értékeiről ad futásidejű, folyamatos tájékoztatást a terminálablakban. Egy komoly terhelés alatt álló adatbázis-kezelő esetében így pontos tájokoztatást kaphatunk az éppen futó lekérdezésekről (és kideríthetjük, mi eszi meg erőforrásainkat).

Az *mtop* futtatásakor a parancssorban legalább két kapcsolót meg kell adnunk:

```
mtop --dbuser=monitor --password=n0telling
```

Értelemszerűen az adatbázisban használt saját felhasználónevünket és jelszavunkat kell megadnunk. Amennyiben a `mtop`-ot nem az adatbázis-kiszolgálóról futtatjuk, a `--host={mysql_`*kiszolgáló*`}` kapcsolóra is szükségünk lesz. A program indulását követően egy néhány másodpercenként frissülő `top`-szerű felületet fogunk látni:

```
load average: 0.72, 0.47, 0.26 mysqld 3.23.51
➥ up 33 day(s), 4:48 hrs
2 threads: 2 running, 0 cached.
➥ Queries/slow: 71.5K/0 Cache Hit: 99.99%
Opened tables: 42 RRN: 4.0M TLW: 0 SFJ: 0 SMP: 0

ID USER HOST DB TIME COMMAND STATE INFO
26049 root localhost test
➥ Query show full processlist
26412 root localhost nocat 1 Query Writing to n
➥ select * from Member where User like '%rob%'
---
```

Máris láthatjuk a kapcsolódott felhasználókat, hogy melyik kiszolgálóról érkeztek, milyen lekérdezést futtatnak (és melyik adatbázison), illetve, hogy melyik szál milyen régóta fut már. A bal oldalon látható szám az egyes szálak azonosítója, nem a MySQL PID érték. Ha egy adott lekérdezés kezd gyaníthatóan lassú lekérdezéssé lenni, akkor az a sor lila színre vált. Amennyiben valóban lassú lekérdezéssé válik, a sor színe sárga lesz. Végül, ha a lekérdezés két MySQL `long_query_time` érték után még fut, a sor vörös színre változik. Így első ránézésre könnyen el lehet dönteni, melyik lekérdezések fogyasztanak különösen sok időt.

Akárcsak a `top` esetében, most is a ? billentyű leütésével juthatunk a kiadható billentyűparancsok listájához:

```
mtop ver 0.6.2/2002905, Copyright (c) 2002,
➥ Marc Prewitt/Chelsea Networks
q - quit
? - help; show this text
f - flush status
k - kill processes; send a kill to a list of ids
s - change the number of seconds to delay between
    ➥ updates
m - toggle manual refresh mode on/off
d - filter display with regular expression
    ➥ (user/host/db/command/state/info)
h - display process for only one host
u - display process for only one user
i - toggle all/non-Sleeping process display
o - reverse the sort order
e - explain a process; show query optimizer info
t - show mysqld stats (show status/mysqladmin ext)
v - show mysqld variables
    ➥ (show variables/mysqladmin vars)
z - zoom in on a process, show sql statement
    ➥ detail
r - show replication status for master/slaves
```

Talán a leggyakrabban használt két parancs az `explain` (e), azaz kifejtés és a `kill` (k), azaz leállítás. Ez e leütése után be kell írnunk a szál azonosítóját. Gépeljük be a kívánt azonosítót, és máris részletesen láthatjuk, mit is csinál pontosan a MySQL a lekérdezés futtatása közben:

```
Id: 27134 User: root Host: localhost Db:
➥ nocat Time: 0
Command: Query State: cleaning up

select *
FROM Member
WHERE User like '%rob%'

table |type |possible_keys |key | ken_len|ref |
➥ rows|
Member |ALL | | | | | 9652|where used
```

Hasonlóképpen a k paranccsal a szálazonosító megadása után leállíthatjuk a folyamatot. Nagyon hasznos dolog, hogy a *mysqladmin* folytonos indítgatása nélkül is le tudjuk állítani a hosszan futó (vagy erőforrás-igényes, rosszul javított) folyamatainkat.

Ha el akarjuk kerülni a felhasználó és a jelszó megadását, létrehozhatunk egy korlátozott jogosultságokkal bíró mysqltop felhasználót. A *perldoc mtop* szerint:

*„A rendszerünket a legkényelmesebben úgy állíthatjuk be az mtop használatára, ha létrehozunk egy mysqltop nevű felhasználót, aki nem rendelkezik jelszóval. Biztonsági okokból e felhasználó minden jogosultságát* `N`*-re kell állítani, kivéve a* `Process_priv` *jogosultságot, ennek értéke Y."*

A jogosultságok megadásához a következő módosításokat hajtsuk végre a MySQL parancssorból:

```
mysql> grant select on test.* to mysqltop;
mysql> grant select on test.* to
mysqltop@localhost;
mysql> update user set process_priv='y'
➥ where user='mysqltop';
mysql> flush privileges;
```

Amennyiben feltelepítjük a *mtop*-ot és beillesztjük a `PATH`-ba, rengeteg gépeléstől (és találgatástól) kímélhetjük meg magunkat, és többé nem kell kizárólag a *mysqladmin* segítségével nyomoznunk a rosszul viselkedő szálak után.

## Kapcsolódó anyagok

- Az *mtop* csomag elérhetősége: `http://mtop.sourceforge.net/` (Perl 5 támogatásra van szüksége.)
- `Curses.pm` (CPAN alatt érhető el.)

# #82 Másolatkészítés MySQL-lel

## Növeljük a teljesítményt és az ismétlésszámot adatbázisunk folyamatosan frissülő másolataival

A 2.23.33-as változattól kezdve a MySQL támogatja az adatbázis-másolatok (database replication) készítését. Ezt a feladatot úgy oldották meg, hogy az egyik gép (az elsődleges) folyamatosan bináris naplót készít az adatbázis-műveletekről, amit a többi (másodlagos) gép saját másolatainak frissítésére használhat. A MySQL 3.23-as változatáig bezárólag ez a másolatkészítés egyirányú (azaz az elsődleges gép adatbázisában történő változások szétterjednek a másodlagos gépeken, de a másodlagos gépeken végzett változások nem kerülnek vissza az elsődlegesre). Amennyiben kétirányú másolatkészítést szeretnénk, vegyük szemügyre a legfrissebb MySQL 4 kódját.

A másolatkészítés leggyakoribb célja, hogy az adatbázis-lekérdezések okozta terhelést több gép között osszuk meg. Ez nemcsak a teljesítményt növeli, de bizonyos megbízhatóság-növekedést is jelent, hiszen bármikor leállhat valamelyik adatbázis-kiszolgáló. Ügyelnünk kell rá, hogy az alkalmazásaink az összes írási feladatot az elsődleges másolaton hajtsák végre, hiszen különben a másolataink nem lesznek összehangoltak. Nyilván mindenki tisztában van vele, hogy ez nem éppen jó dolog, ezért mindig győződjünk meg róla, hogy kódjainkban az összes írási műveletet kizárólag az elsődleges másolaton végezzük el.

Kilenc egyszerű lépésben foglaltuk össze, hogyan lehet másolatkészítést létrehozni a MySQL 3.23.33-as (vagy későbbi) változat alatt. Mielőtt bármihez hozzáfognánk, ellenőrizzük, hogy valamennyi gépen azonos MySQL-változatot használunk-e (lehetőleg a legfrissebb, üzembiztos változatot).

1. Döntsük el, melyik gép lesz az elsődleges, és melyek a másodlagosak. Általában elsődleges gépnek a legerősebbet érdemes választani. Amennyiben egy népszerű oldalt üzemeltetünk, amelynek számos adatbázis-kérelmet kell kezelnie, érdemes elgondolkodnunk egy beépített RAID-szolgáltatással és sok memóriával rendelkező többprocesszoros gép beszerzésén. Ha az adatbázis-kiszolgálónk alulméretezett, a teljes dinamikus alkalmazásunkat

súlytani fogja a teljesítménycsökkenés, mivel a többi kiszolgáló a terhelés elosztására használt gépek számától függetlenül, üresjáratban fog várakozni az adatbázis frissítésére.

2. Készítsünk az elsődleges kiszolgálón egy replicant (másolat) nevű felhasználót. Ez a felhasználó semmiben sem fog különbözni a többiektől, eltekintve attól, hogy az összes másolandó adatbázishoz `FILE` jogosultságokat kap.

```
mysql> grant FILE on webdb.* to
➥ replicant@'%.mynetwork.edu' identified by
➥ 'seCret';
```

3. Az elsődlegesen engedélyezzük a *binlog* szolgáltatást, és válasszuk ki mely adatbázisokat fogjuk másolni. Illesszük az elsődleges */etc/my.cnf* állományának `[mysqld]` részébe a következőket:

```
log-bin
server-id=1
```

Továbbá jegyezzünk be minden másolandó adatbázist a `binlog-do-db` kulcsszóval:

```
binlog-do-db=webdb
```

A változtatások érvényesítése érdekében állítsuk le, majd azonnal indítsuk is újra a MySQL-kiszolgálót:

```
root@db:/usr/local/mysql# mysqladmin shutdown;
➥ ./bin/safe_mysqld&
```

4. Állítsuk le az elsődleges kiszolgáló adatbázis-kezelőjét, és készítsünk másolatot a *data/* könyvtárról. Figyelem, a jelenlegi *mysqldump* nem felel meg a célnak, nekünk a *data/* könyvtár tényleges másolatára van szükségünk. Ez néhány másodperctől kezdve akár perceket is igénybe vehet, attól függően mennyi adatot tartunk a kiszolgálón:

```
root@db:/usr/local/mysql# mysqladmin shutdown;
➥ tar cvf ~/data.tar data/
```

Adatbázisunk sajnos elérhetetlen lesz a másolatkészítés teljes időtartama alatt, így ügyeljünk rá, hogy ne a forgalmas órákban indítsuk el a műveletet. Egy kis szerencsével csak egyszer kell végigjátszanunk.

5. Várjuk meg míg az elsődleges kiszolgáló adatbázis-másolata elkészül. Ha úgy tűnik, hogy a *tar* hiba nélkül lefutott, indítsuk újra a MySQL-kiszolgálót:

```
root@db:/usr/local/mysql# ./bin/safe_mysqld&
```

6. Győződjünk meg róla, hogy a másodlagos kiszolgálókon nem fut a MySQL, majd az adatokat másoljuk át rájuk. A másodlagos kiszolgálókon adjuk meg a következőt:

```
root@slave:~# mysqladmin shutdown; scp
➥ db:data.tar .
```

7. Válasszunk minden másodlagos kiszolgálónak egyedi azonosítót, majd indítsuk be a másolatkészítést. Az összes másodlagos kiszolgáló */etc/my.conf* állományának `[mysqld]` szakaszába jegyezzük be a következő sorokat:

```
master-host=db.mynetwork.edu
master-user=replicant
master-password=seCret
server-id=10
```

Ügyeljünk rá, hogy a `server-id` értéke minden másodlagos kiszolgálón egyedi legyen. Bármilyen egész számot választhatunk, amit még nem használtunk fel egy másik másodlagos kiszolgálóhoz (vagy az elsődlegeshez). Amennyiben a másodlagos gép kizárólag az elsődleges kiszolgáló „szolgájaként" fog üzemelni (azaz nem fog önmagában saját írható, illetve olvasható adatbázisokat kezelni) további teljesítménynövekedést érhetünk el a következő sorok beiktatásával:

```
low-priority-updates
skip-bdb
delay-key-write-for-all-tables
```

A fenti sorokkal elkerülhetjük a felesleges kódok használatát, amelyeket egyébként csak az adatbázis írási műveleteihez alkalmaznánk. Másodlagos kiszolgálóként üzemelve soha nem lesz szükségünk ténylegesen adatbázis-írásra, így a MySQL ilyen képességeire sincs szükségünk.

8. Az összes másodlagos kiszolgálón tömörítsük az adatokat:

```
root@slave:/usr/local/mysql# mv data data.old;
➥ tar vxf ~/data.tar
```

A frissen létrejött *data* könyvtár tulajdonosa a mysql felhasználó és csoport legyen:

```
root@slave:/usr/local/mysql#
➥ chown -R mysql.mysql data/
```

Ha nincs szükségünk a másodlagos kiszolgáló eredeti *data* könyvtárára, nyugodtan le is törölhetjük. (Nem muszáj elmenteni *data.old* néven.)

9. Indítsuk el az összes másodlagos kiszolgálón a MySQL-t és figyeljük a hibanaplókat. Indítsuk be a másodlagos kiszolgálók MySQL-kiszolgálóit:

```
root@slave:/usr/local/mysql# ./bin/safe_mysqld&
```

és figyeljük a naplókat:

```
root@slave:/usr/local/mysql#
➥ tail -f data/slave.err
```

(Természetesen a másodlagos helyére a valódi gépnevet kell írni.) Amennyiben valami ilyesmi üzentet látunk: *„Starting replication at position XYZ"*, akkor jó hírünk van, a másolatkészítés működik. Végezzünk valami változtatást az elsődleges kiszolgálón és kérdezzük le a másodlagoson, hogy lássuk megfelelően terjed-e az adat. A frissítés általában közel azonnali. Amennyiben a másolatkészítéssel valami gond adódik, az üzenetek a másodlagos kiszolgálók mysql naplófájljaiban fognak megjelenni.

A másolatkészítés során felbukkanó nehézségek többnyire a jogosultságokkal kapcsolatosak (ellenőrizzük a 2. pontban megadott GRANT paraméterezését), illetve előfordulhat, hogy a másodlagos kiszolgálók adatállománya nincs összhangban az elsődleges kiszolgálóval (ismételjük meg a 4., 5., 6., 8. és 9. lépéseket *nagyon* gondosan). Haladjunk megfontoltan, és mindig nézzük meg minden gép MySQL hibanaplóit.

Amennyiben később további másodlagos kiszolgálókat szeretnénk üzembe állítani, tartsuk kéznél a *data.tar* állományunk egy másolatát. Segítségével bármikor új másodlagos kiszolgálókat tudunk készíteni (a másodlagos hálózaton keresztül egyeztetni fog a elsődlegessel, visszajátszva a *data.tar* készítése óta folyamatosan bővülő *binlog* naplót).

Másolatkezelést létrehozni és karbantartani ezeket az adatbázis-másolatokat két egészen különböző feladat. Amíg az alkalmazásunk elég okos ahhoz, hogy csak az elsődleges kiszolgálón módosítson, és amíg az alkatrészeink megbízhatóak, általában nemigen ütközünk gondokba. De ha egyszer a másodlagos kiszolgálóink kiesnek az összhangból, akkor sokkal több adatra lesz szükségünk, mint amit ebben az egyszerű fejezetben le tudok írni. Mielőtt még bármilyen összetettségű másolatkészítő telepítésébe kezdenénk, mindenképpen olvassuk el az alább felsorolt kitűnő forrásmunkákat.

### Kapcsolódó anyagok

- MySQL Reference Manual (O' Reilly)
- MySQL hálózati dokumentáció
  `http://www.mysql.com/doc/en/Replication.html`

## #83 Hogyan hozzunk vissza nagyméretű MySQL-mentésekből egyetlen táblát

### Egy módszer, amellyel a hatalmas MySQL-mentésekből is visszahozhatjuk bármelyik táblát

Mint minden jó rendszergazda, nyilván mi is minden este elmentjük a MySQL tábláinkat, majd tömörített formában lemezre másoljuk őket (valószínűleg egy idő után mindezt parancsfájlok segítségével végezzük). Elképzelhető, hogy adatbázis-kiszolgálónk (vagy az egyik másodlagos kiszolgáló) `cron` fájljába bejegyeztünk valami ilyesmit:

```
for x in `mysql -Bse show databases`; do
mysqldump $x | gzip -9 >
➥ /var/spool/mysqldump/$x.`date +%Y%m%d`.gz
done
```

Így biztonságban lehetünk még akkor is, ha az élő adatbázissal valami végzetes dolog történne. Azonban amikor az adatbázisunk már kezd figyelemre méltó méreteket ölteni, a részleges adatbázis-visszaállítás nem is olyan egyszerű. A néhány millió soros adatbázisoknál a mentések kezdenek hatalmas adathalmazokká válni. Hogyan állíthatunk vissza egyetlen táblát egy ilyen több száz megabájtos tömörített mentésből?

Lássunk egy egyszerű módszert Perl nyelven. Készítsük el az *extract-table* nevű parancsfájlt az alábbi tartalommal:

```
#!/usr/bin/perl -wn
BEGIN {  $table = shift @ARGV }
print if /^create table $table\ b/io ..
➥ /^create table (?!$table)\ b/io;
```

Ha például a `User` táblára van szükségünk a randomdb nevű adatbázisból, a következő utasítást adjuk ki:

```
# zcat /var/spool/mysqldump/randomdb.20020901.gz |
➥ extract-table Users > ~/Users.dump
```

Így már a következő egyszerű paranccsal vissza tudjuk állítani a `User` táblánkat:

```
# mysql randomdb -e "drop table Users"
# mysql randomdb < ~/Users.dump
```

## #84 MySQL-kiszolgáló felpörgetése

### Próbáljuk ki a következő célszerű módosításokat, hogy a lehető leghatékonyabban használjuk MySQL-kiszolgálónkat

Gyakran megesik, hogy nincs senki más, aki el akarná (vagy el tudná) végezni ezt a feladatot, s így a rendszergazda nyakába szakad a DBA (DataBase Administrator, azaz adatbázis-rendszerfelügyelő) felelőssége is. Számos ember egész életét adatbázisok finomításával és karbantartásával tölti, de soha rá se néz a rendfelügyeleti feladatokra. Mégis nem egy Linux-rendszergazdát láttam már, akinek alig valami gyakorlattal el kellett vállalnia a DBA-feladatkört is („természetesen" fizetésemelés nél-

kül). Bár a következő módosítás nem fog egyből DBA-szakemberré változtatni senkit, azért reményeim szerint lesz benne egy-két olyan célszerű lépés, amely segít a gyakorlati élet adatbázisainak felpörgetésében.

Következzen tehát öt MySQL adatbázis-teljesítményjavító lépés közelítőleg nehézségi (és hatékonysági) sorrendben:

1. Adjuk ki a `mysqlcheck -o` adatbázis-parancsot. Ezzel hatékonyabbá tehetjük (optimize) a tábláinkat, és az adatbázis „töredezettség-mentesítésével" újra munkába állíthatjuk az elveszett helyeket. Különösen akkor hasznos megoldás, ha mostanában változtattuk meg az adatbázis szerkezetét, vagy nagyobb mennyiségű adatot töröltünk belőle.
2. Adjunk új fontosságot (priority) a `mysqld`-nek. Amennyiben kinevezett (dedicated) MySQL-kiszolgálóval rendelkezünk, megadhatjuk az ütemezőnek, hogy a MySQL-t minden más folyamatnál sokkal magasabb fontossági szinten futtassa. A MySQL kézikönyv javaslata szerint a következő sort érdemes a `safe_mysql` parancsfájlba írni:

   ```
   renice -20 $$
   ```

   Ugyanakkor nekem úgy tűnt, hogy nem árt kikeresni a következő méretes kódrészletet:

   ```
   NOHUP_NICENESS="nohup"
   if test -w /
   then
   NOHUP_NICENESS=`nohup nice 2>&1`s
   if test $? -eq 0 && test x"$NOHUP_NICENESS"
   ➥ != x0 && nice --1 echo foo > /dev/null 2>&1
   then
   NOHUP_NICENESS="nice --$NOHUP_NICENESS nohup"
   else
   NOHUP_NICENESS="nohup"
   fi
   fi
   ```

   és lecserélni a következő sorra:

   ```
   NOHUP_NICENESS="nohup nice --20"
   ```

Ettől kezdve a `safe_mysql` és az összes `mysqld` folyamat a lehető legnagyobb fontossággal fut. Ezt a csak a többi folyamat rovására teheti meg, így ha más szolgáltatást is akarunk a MySQL mellett futtatni, érdemes magasabb számot választani (valahol a –10-től a –5-ig tartományban).

3. Készítsünk indexeket. Amennyiben hosszú futású lekérdezésekkel dolgozunk, kitűnő hatékonyságnövelési lehetőség, ha indexeket adunk a tábláinkhoz. Ha egy hosszan futó lekérdezést látunk az *mtop* használata során (erről a [#81] „MySQL felügyelete az mtop segítségével" trükkben olvashattunk bővebben) nézzük meg, hátha tudunk indexekkel segíteni a dolgon:

```
mysql> create index name on Member (Name(10));
```

Az index végső soron lemezterület feláldozása a teljesítménynövelés érdekében, így manapság az olcsó háttértárak korában, szinte megbocsáthatatlan, ha elhanyagoljuk az indexeket. Általában nem származik nagy gond abból, ha túl sok indexet készítünk, de soros keresést (vagy egyedi beillesztést) végezni egy nagy, indexeletlen táblán iszonyatosan lelassíthatja a kiszolgálónkat.

4. Ellenőrizzük a kiszolgáló változóit. Az alapértelmezett változókat és biztonságot szem előtt tartva, átlagos felszereltségű gépekhez tervezték. Ha komolyabb mennyiségű memóriával rendelkezünk (512 MB vagy még több) nagy hasznát láthatjuk, ha feljebb vesszük az alapértelmezett gyorstár- és veremméreteket. Az alábbi beállításokat egy üzleti adatbázis-kiszolgálón használtuk (kétprocesszoros Pentium 4, 1,0 GHz, 2 GB memória és rengeteg gyorstár lemezterület). A sorokat a */etc/my.cnf* állomány `[mysqld]` szakaszába kell bejegyezni:

```
set-variable = key_buffer=384M
set-variable = max_allowed_packet=1M
set-variable = table_cache=512
set-variable = sort_buffer=2M
set-variable = record_buffer=2M
set-variable = myisam_sort_buffer_size=64M
set-variable = tmp_table_size=8M

set-variable = max_connections=768
```

Az igazsághoz hozzátartozik, hogy a fentiek nagy részét egyenesen a MySQL-terjesztéssel együtt érkező *my-huge.cnf* példából ollóztuk ki (nem nagyon kellett megváltoztatnunk őket, mivel nagyszerűen működtek a rendszerünkkel). Mivel a webkiszolgálónkon `Apache::DBI`-t futtatunk, a `wait_timeout` értéket néhány percre állítottuk:

```
set-variable = wait_timeout=120
```

Így a függőben lévő DBI-szálak nem maradnak meg örökké, és nem fogyasztják el a *max_connections* jellemzőben megadott összes elérhető helyet.

5. Foltozzuk meg a glibc-t és a szálakat. Ha már az alapértelmezett glibc összes lehetőségét kimerítettük, esetleg elgondolkozhatunk a foltozáson (lásd még a [#86] „Készítsünk szuper MySQL-kiszolgálót: tegyük hatékonnyá a rendszermagot, a glibc-t és a Linux-szálakat" trükköt), hogy kisebb szálakat és több nyitott fájlt kapjunk. Általában ilyesmi csak a legnagyobb, legleterheltebb MySQL-telepítések esetében jöhet szóba.
Mint azt már lassan megszokhattuk, az adatbázisok javítása, egyszerűsítése és karbantartása is jóval összetettebb feladat annál, mintsem e néhány lapon egészében lefedhessük. Az alábbi forrásokban további, a témával kapcsolatos, hasznos útmutatókra lelhetünk.

### Kapcsolódó anyagok

- MySQL Refernce Manual (O' Reilly)
- MySQL (New Riders)
- `http://www.mysql.com/doc/en/Linux.html`
- `http://www.mysql.com/doc/en/SHOW_VARIABLES.html`

## #85 Használjunk MySQL-t proftpd-hez azonosítási forrásként

### A proftpd és a MySQL segítségével többé nem lesz szükségünk külön bejelentkezés-azonosítókra minden egyes FTP-felhasználóhoz

A `proftpd` FTP-démon az Apache kiszolgáló formátumához nagyon hasonló beállítással rendelkező, igen hatékony rendszer. Egy egész rakás különleges lehetőséget tartogat, amelyet máshol nemigen talá-

lunk meg. Támogatja az arányokat, virtuális gazdák készítését, ezenkívül moduláris felépítésű, így saját magunk is kiegészíthetjük további modulokkal.

Az egyik ilyen modul a `mod_sql`, melynek segítségével a `proftpd` képes adatbázisokat használni azonosítási forrásként. A `mod_sql` jelenleg a MySQL és a PostgreSQL rendszereket támogatja. Rendszerünk bejáratainak lezárására kitűnő ez a módszer, hiszen a belépő felhasználókat az adatbázison keresztül azonosíthatjuk (ekként nem lesz szükség felhasználónkénti külön héjprogram-azonosítókra). Ebben a módosításban bemutatjuk, hogyan azonosíthatjuk a `proftpd` vendégeit MySQL adatbázison keresztül.

Első lépéként töltsük le és telepítsük a `proftpd` és a `mod_sql` forrását:

```
~$ bzcat proftpd-1.2.6.tar.bz2 | tar xf -
~/proftpd-1.2.6/contrib$ tar zvxf
➥ ../../mod_sql-4.08.tar.gz
~/proftpd-1.2.6/contrib$ cd ..
~/proftpd-1.2.6$ ./configure
➥ --with-modules=mod_sql:mod_sql_mysql \
--with-includes=/usr/local/mysql/include/ \
--with-libraries=/usr/local/mysql/lib/
```

(Természetesen amennyiben MySQL példányunk elérési útvonala eltérne a */usr/local/mysql* útvonaltól, akkor helyette azt kell begépelnünk.) Építsük össze a kódot és telepítsük:

```
rob@catlin:~/proftpd-1.2.6$
➥ make && sudo make install
```

Készítsünk új adatbázist a `proftpd`-nek (tételezzük fel, hogy már van egy működő MySQL rendszerünk):

```
$ mysqladmin create proftpd
```

Ezt követően engedélyezzük a `proftpd` csak olvasható elérését:

```
$ mysql -e "grant select on proftpd.*
➥ to proftpd@localhost \
    identified by 'secret';"
```

Készítsünk táblát az alábbi táblaleírások alapján:

```
CREATE TABLE users (
userid varchar(30) NOT NULL default '',
password varchar(30) NOT NULL default '',
uid int(11) default NULL,
gid int(11) default NULL,
homedir varchar(255) default NULL,
shell varchar(255) default NULL,
UNIQUE KEY uid (uid),
UNIQUE KEY userid (userid)
) TYPE=MyISAM;

CREATE TABLE groups (
groupname varchar(30) NOT NULL default '',
gid int(11) NOT NULL default '0',
members varchar(255) default NULL
) TYPE=MyISAM;
```

A táblák elkészítésének legegyszerűbb módja, ha a fenti sorokat mentjük egy *proftpd.schema* nevű fájlba, majd kiadjuk a `mysql proftpd < proftpd.schema` parancsot.

Ideje megmutatnunk a `proftpd`-nek, melyik adatbázist használhatja az azonosításhoz. A következő sorokat adjuk a */usr/local/etc/proftpd.conf* állományhoz:

```
SQLConnectInfo proftpd proftpd secret
SQLAuthTypes crypt backend
SQLMinUserGID 111
SQLMinUserUID 111
```

Az `SQLConnectInfo` formátuma az adatbázis, felhasználó, jelszó sorrendet követi. Akár egy másik gépen (esetleg másik kapun) keresztül elérhető adatbázist is megadhatunk a következő formában:

```
SQLConnectInfo
➥ proftpd@dbhost:5678 valaki valamilyenjelszó
```

Az `SQLAuthTypes` sorral adhatjuk meg, hogy a jelszavakat szabályos Unix crypt formátumban tároljuk, avagy a MySQL `PASSWORD()` függ-

vényét használjuk. Figyelem, amennyiben használjuk a `mod_sql` naplózó képességeit, védjük a naplófájlokat, mert a jelszavak nyílt szöveges formában is megjelenhetnek bennük.

Az `SQLAuthTypes` fenti formájában nem engedélyezi az üres jelszavakat, ha ilyesmit szeretnénk, egészítsük ki az `empty` kulcsszóval. A `SQLMinUserGID` és `SQLMinUserUID` kulcsszavakkal azt a legkisebb felhasználói és csoportazonosítót adhatjuk meg, amelyet a `proftpd` még hajlandó beengedni. Nem árt, ha ezt az értéket nullánál nagyobb értékre állítjuk (ezzel letiltva a rendszergazdai bejelentkezést), de csak annyira legyen kicsi, hogy a fájlrendszer megfelelő jogosultságait még ki tudjuk osztani. Ezen a rendszeren létezik egy www nevű felhasználónk és csoportunk, ezek azonosítója egyaránt 111. Mivel szeretnénk, ha a webfejlesztők ezekkel a jogosultságokkal lépnének be, minimumértékként ezt kell beállítanunk.

Végül készen állunk, hogy az adatbázisban elhelyezzük a felhasználókat. Az alábbi sor létrehozza a jimbo nevű felhasználót, akinek a felhasználói jogosultsága `www/www` lesz és bejelentkezéskor a */usr/local/apache/htdocs/* könyvtárba fog kerülni:

```
mysql -e "insert into users values
('jimbo',PASSWORD('blabla'),'111', \
  '111', '/usr/local/apache/htdocs','/bin/bash');"
proftpd
```

A jimbo felhasználó jelszavát tárolás előtt a MySQL `PASSWORD()` függvénye titkosítja. A */bin/bash* sor csak azért adjuk át a `proftpd`-nek, hogy teljesítsük a *proftpd* `RequireValidShell` követelményét, és természtesen szó sincs arról, hogy a jimbo felhasználó bármiféle héjprogram-hozzáférést kapna.

Ezek után beindíthatjuk a `proftpd`-t, és a blabla jelszóval máris be tudunk jelentkezni jimbóként. Amennyiben gondok merülnének fel a bejelentkezés során, futtassuk a `proftpd`-t az előtérben és kapcsoljuk be a hibakeresést:

```
# proftpd -n -d 5
```

Figyeljük a megjelenő üzeneteket a bejelentkezés alatt, és remélhetőleg megtaláljuk a hiba forrását. Tapasztalataim szerint a legtöbb esetben a hibás *proftpd.conf* a ludas, általában valamilyen jogosultsági okból kifolyólag.

A `mod_sql` az itt bemutatottaknál sokkal többre képes; létező adatbázisok tetszőleges táblájához is hozzá tud kapcsolódni, képes az összes műveletet az adatbázisba naplózni, vagy szűrni a felhasználókat tetszőleges `WHERE` feltétel szerint.

### Kapcsolódó anyagok

- A `mod_sql` honlapjának címe:
  `http://www.lastditcheffort.org/~aah/proftpd/mod_sql/`
- A `proftpd` honlapja: `http://www.proftpd.org/`

## #86 Készítsünk szuper MySQL-kiszolgálót: a glibc, a Linuxthreads és a rendszermag finomhangolása

### E módosításokkal elérhetjük, hogy adatbázis-kezelőnk operációs rendszere a lehető leghatékonyabban fusson

Amennyiben erősen leterhelt MySQL-kiszolgálót futtatunk (mondjuk 800 feletti lekérdezést kapunk másodpercenként, és ügyfelek százai lógnak rajta folyamatosan), előfordulhat, hogy az operációs rendszer korlátaiba ütközünk, melyek miatt a MySQL már egyszerűen nem tud elég hatékonyan működni. Szélsőséges esetben (rosszul viselkedő szálkezelő könyvtárral rendelkező gépeken) megtörténhet, hogy a kényes erőforráshatárt elérve a MySQL egyszer csak az összes processzoridőt elfogyasztja, mesterségesen akár 100-as érték fölé túlhajtva a terhelést (load).

Ha azonban külön MySQL-kiszolgálót állítunk fel, módosítjuk a glibc néhány alapértelmezett értékét, átírjuk a Linux-szálkezelést és a Linux-rendszermagot, akkor akár olyan rendszert is létrehozhatunk, amely egyetlen gépen kérelmek ezreit képes egyszerre kezelni. Természetesen olyan alkatrészekre lesz szükségünk, amelyek képesek

elviselni ezt a terhelést, de a módosítások elvégzése után remélhetőleg többé már nem az operációs rendszer és a MySQL-változatunk lesz a teljesítmény korlátja.

*Figyelem, az itt olvasható módosítások a kiszolgáló létfontosságú, érzékeny részeit érintik. Ne játszadozzunk felelőtlenül a libc és rendszermag forrásokkal, és mindenképpen csak azután vágjunk bele a módosításokba, hogy elkészítettük a teljes rendszer mindenre kiterjedő, ellenőrzött, hálózaton (és lehetőleg gépen) kívüli biztonsági másolatát. A változtatások elvégzését kizárólag kinevezett MySQL gépekhez javasoljuk, és csak akkor, ha teljesen biztosak vagyunk benne, hogy minden más rendben működik (különös tekintettel a /etc/my.conf változók beállításaira). Mi figyelmeztettünk!*

### 1. lépés: glib felépítése

Töltsük le a glibc forrását a `http://www.gnu.org/software/libc/libc.html` címről. Az írás születésekor a legfrissebb változat a glibc 2.2.5. Mindjárt töltsük le glibc változatunkhoz tartozó `linuxthreads` állományt is.

Csomagoljuk ki a *glibc* állományt, majd telepítsük a `linuxthreads` modult (a glibc terjesztésben leírtaknak megfelelően). Ezután kövessük a `http://www.mysql.com/doc/en/Linux.html` címen olvasható útmutatásban foglaltakat a következőképpen:

- állítsuk a *sysdeps/unix/sysv/linux/bits/local_lim.h* állomány `PTHREAD_THREADS_MAX` értékét 4096-ra.
- állítsuk a *linuxthreads/internals.h* `STACK_SIZE` értékét 256 KB-ra.

A `configure` futtatása során adjuk meg a következő kapcsolót: `--prefix=/usr/local/glibc-2.2.5`. Így a glibc a saját könyvtárban épül fel, és nem fogja lecserélni a rendszer saját glibc állományát. Ez nagyon fontos, mivel hogyha a `make install` begépelésével felülírjuk a futó rendszer glibc állományát, általában teljesen összekavarjuk a programkönyvtárat! A tönkrement libc-t pedig igen nehéz megjavítani statikusan csatolt futtatható rendszereszközök és rendszerlemez nélkül, ezért nem is javaslom a kipróbálását, feltéve, hogy nincs rengeteg felesleges időnk (és egy teljes másolatunk az ép-

pen tönkretett gép állományairól). Mivel mi a glibc-t egy másik könyvtárba telepítjük, ezzel ügyesen megkerüljük a rendszer libc frissítésével kapcsolatos egész galibát.

Fordítsuk le és telepítsük a glibc-t. Ha minden jól megy, a */usr/local/glibc-2.2.5/* könyvtár alatt hamarosan egy csillogó új glibc-vel büszkélkedhetünk.

### 2. lépés: a rendszermag

A rendszermagfán a következő változtatásokat végezzük el:

- *include/linux/limits.h*: állítsuk a `NR_OPEN` értéket 4096-ra
- *include/linux/fs.h*: állítsuk a `INR_OPEN` értéket 4096-ra

Fordítsuk újra a rendszermagot (és a modulokat), telepítsük az új rendszermagot, majd indítsuk újra a gépet.

### 3. lépés: új MySQL készítése

Töltsük le és telepítsük a MySQL forrásfát. A `configure` futtatásakor adjuk meg a `--use-other-libc=/usr/local/glibc-2.2.5` kapcsolót. A MySQL ezáltal a mi új glibc könyvtárunkat használja majd a rendszer glibc helyett. Fordítsuk le és telepítsük a MySQL-t a szokásos módon.

### 4. lépés: rendszerindításkor használjuk a lehető legtöbb fájlkezelőt

Adjuk a következő sort a */etc/rc.d/rc.local* fájlhoz (vagy a rendszerbehúzó folyamat bármely más részéhez, amely a `safe_mysqld` előtt fut le):

```
echo 65536 > /proc/sys/fs/file-max
```

Indítsuk újra a rendszert (vagy írjuk be kézzel a fenti sort) és indítsuk be a `safe_mysql`-t. Házon belül futtatott teljesítményméréseink szerint az elérhető kapcsolatok legnagyobb száma minden látható processzorteljesítmény-növekedés nélkül néhány százról 4090-re ugrott. Ezekkel a beállításokkal futtatunk több kinevezett üzleti adatbázis-kiszolgálót, és eddig még semmiféle teljesítmény vagy megbízhatósági gondot nem tapasztaltunk.

Amennyiben jelentős mértékű adatbázisforgalmat kell lebonyolítanunk, érdemes elgondolkodni a terhelés szétosztásán. Erről a [#82] „Másolatkészítés MySQL-lel" trükkben írtunk.

## #87 Apache Toolbox

### Ezzel a nagyszerű eszközzel könnyedén letölthetjük, beállíthatjuk, lefordíthatjuk és telepíthetjük az Apache-t (és társait)

A Bryan Andrew által írt Apache Toolbox igazi svájcibicska, amely testreszabható, menüvezérelt felületet kínál az Apache, `mod_perl`, MySQL, PHP és még sok más hasznos dolog letöltéséhez és telepítéséhez. Az Apache Toolbox a következő programokat támogatja:

Apache
: mindenki kedvenc webkiszolgálója,

SSL
: Secure Socket Layer a biztonságos webkiszolgáló műveletekhez,

PHP, `mod_perl` és `mod_fastcgi`
: a gyors parancsfájltámogatás kedvéért,

MySQL
: a mindenhol megjelenő szélsebes adatbázis-kezelő,

OpenLDAP, `mod_auth_ldap`, `mod_auth_radius`, `mod_auth_pop3`, `mod_auth_sys`, `mod_accessref`
: különböző eljárások a hitelesítés és jogosultság-ellenőrzés világából,

WebDAV és `mod_layout`
: egyszerű, ám hatékony webtervező eszközök,

`mod_dynvhost`, `mod_throttle`, `mod_gzip` és `mod_bandwidth`
: a hatékony háttérteremtés és kiszolgáló-irányítás kedvéért.

És ez még nem minden! Az Apache Toolbox testreszabható és szinte bármit képes támogatni, amit csak bele akarunk illeszteni (hiszen végső soron egy parancsfájlról van szó).

Az Apache Toolbox két különböző változatban létezik: csupasz parancsfájl formájában, amely igény szerint tölti le a különböző összete-

vők forrásait, illetve teljes csomagként, amelyben a parancsfájl mellett egyúttal az összes forrás is megtalálható. Az eszközkészlet még a fennálló RPM ütközéseket is képes észrevenni!

Maga az Apache Toolbox tulajdonképpen egy parancsfájl, amelyet a parancssorból futtathatunk. Rendszergazdaként kell végrehajtani (mint arra emlékeztet is), hogy az összes alkotórészt a megfelelő helyre illeszthesse. Elsőként a menüvezérelt felülettel találkozunk, ahol kiválaszthatjuk, pontosan mely csomagokat szeretnénk telepíteni:

```
-------------------------------------------------
Apache Toolbox 1.5.59
Support: http://www.apachetoolbox.com
-------------------------------------------------
[+] apache) Apache submenu...
[-] php) PHP submenu (v4.2.2)...
[-] rpm) Build an RPM with your choices?
[-] page2) Apache Modules PAGE 2 ...
-------------------------------------------------
[-] 1) GD 2.0.1 [-] 2) -SQL DB Menus-
[-] 4) Mod Python 2.7.8 [-] 5) Mod_SSL+OpenSSL
[-] 6) -Mod Throttle 312 [-] 7) -WebDAV 1.0.3-1.3.6
[-] 8) -Mod FastCGI [-] 9) -Mod AuthNDS 0.5
[-] 10) -Frontpage 2002 [-] 11) -Mod GZIP 1.3.19.1a
[-] 12) -Mod DynaVHost [-] 13) -Mod Roaming
[-] 14) -Mod AccessRef 1.0.1 [-] 15) -Mod AuthSYS
[-] 16) -Mod Bandwidth [-] 17) -Mod Perl 1.27
[-] 18) -Mod Auth LDAP [-] 19) -Apache Jakarta
[-] 20) -Mod Auth Radius [-] 21) -Mod Auth POP3
[-] 22) -Mod Layout 3.2 [-] 23) -Mod DTCL
q) Quit 99) Descriptions
go) Compile selections...
-------------------------------------------------
```

De vajon mit jelent ez a rengeteg mindenféle? Teljes leírást kaphatunk, ha ütünk egy ENTER-t.

Miután összeállítottuk kívánságlistánkat, gépeljük be a go parancsot és az eszközkészlet máris működésbe lép. Az első állomás az RPM ütközések vizsgálata:

```
-------------------------------------------------
------------ Scanning for RPM's -----------------
-------------------------------------------------
Testing for PHP RPM... not found.
Testing for PHP IMAP RPM... not found.
Testing for GD RPM... not found.
Testing for GD Devel RPM... not found.
Testing for Apache RPM... not found.
...
Testing for OpenLDAP RPM... not found.
Testing for OpenLDAP Devel RPM... not found.
[+] Wget found!
-------------------------------------------------
```

Az eszközkészlet megerősítést kér az Apache telepítési útvonalára *(/usr/local/apache)*, lehetővé téve, hogy tetszés szerint megváltoztassuk a végcélt, majd folytatja a munkát. Amennyiben valamelyik forrást (*tar.gz*) nem találja meg, engedélyt kér a letöltésére:

```
[+] Setting up Apache source...
[-] apache_1.3.26.tar.gz detection failed
Do you wish to download it now? [y/n] y

--21:35:40--
--
ftp://mirrors.partnersforever.net:21/pub/apache/dist/
apache_1.3.26.tar.gz => `apache_1.3.26.tar.gz'
Connecting to mirrors.partnersforever.net:21...
➥ connected!
Logging in as anonymous ... Logged in!
==> TYPE I ... done. ==> CWD pub/apache/dist ...
➥ done.
==> PORT ... done.
➥ ==> RETR apache_1.3.26.tar.gz ... done.
Length: 2,303,147 (unauthoritative)

0K -> ........ ........ ........ ........ [ 2%]
50K -> ........ ........ ........ ........ [ 5%]
...
```

Ezt követően az eszközkészlet csendesen elmolyol a beállításokkal, összeállítással, előkészítéssel és egyéb teendőkkel:

```
[+] Uncompressed Apache source...
[+] Getting apache pre-configured
[+] Apache pre-configured
[+] Apache httpd.conf-dist updated for SSI support
[+] Getting GD lib's with PNG and zlib support
ready...
...
```

Kis idő múltán lehetőségünk nyílik az Apache beállításfájljának szerkesztésére, feltéve, hogy elég merészek vagyunk az ilyesmihez. Ha minden jól ment és már korábban is volt már szerencsénk az Apache-hoz, hamarosan ismerős kép fogad bennünket:

```
Configuring for Apache, Version 1.3.26
+ using installation path layout:
➥ Apache (config.layout)
+ activated php4 module (modules/php4/libphp4.a)
+ Warning: You have enabled the suEXEC feature.
➥ Be aware
+ that you need root privileges to complete
➥ the final
+ installation step.
Creating Makefile
Creating Configuration.apaci in src
Creating Makefile in src
+ configured for Linux platform
...
Creating Makefile in src/modules/standard
Creating Makefile in src/modules/php4
[+] Done Configuring Apache source

-----------------------------------------------
If there where _no_ errors run "cd
apache_1.3.26;make" now.
Start debugging and have a blast...
Run "make install" in the apache source
directory to install apache 1.3.26
-----------------------------------------------
```

Az útmutatásnak megfelelően kotorásszunk kicsit az Apache könyvtárban, és adjuk ki a szokásos utasításokat: `make`, `make test` majd végül `make install`!

### Kapcsolódó anyagok

- Érdemes elolvasni az eredeti cikket a `http://www.onlamp.com/pub/a/apache/2000/11/17/wrangler.html` címen
- Apache Toolbox
- Linux Apache MySQL PHP (LAMP) Guide (Linux Help.Net)
- HTTP Wrangler columns (O'Reilly Network)

## #88 Teljes fájlnév megjelenítése listákban

### Kapcsoljuk ki a fájlnevek rövidítését önműködő könyvtárlistáinkban

Biztosan feltűnt, hogy a könyvtárlistákban az Apache lerövidíti a fájlneveket. Igencsak kiábrándító, mikor egy csupa mókás állománnyal teli kiszolgálóra belépve a következőt látjuk:

```
Index of /~rob/stuff/kernel

Name Last modified Size Description

Parent Directory 03-Sep-2002 00:33 -
patched-linux-2.4.12..> 11-Oct-2001 00:59 22.0M
patched-linux-2.4.12..> 11-Oct-2001 00:59 1k
patched-linux-2.4.12..> 11-Oct-2001 00:59 27.1M
patched-linux-2.4.12..> 11-Oct-2001 00:59 1k
patched-linux-2.4.13..> 23-Oct-2001 22:28 22.0M
patched-linux-2.4.13..> 23-Oct-2001 22:28 1k
patched-linux-2.4.13..> 23-Oct-2001 22:28 27.2M
patched-linux-2.4.13..> 23-Oct-2001 22:28 1k
patched-linux-2.4.14..> 05-Nov-2001 15:30 22.1M
patched-linux-2.4.14..> 05-Nov-2001 15:30 1k
patched-linux-2.4.14..> 05-Nov-2001 15:30 27.4M
patched-linux-2.4.14..> 05-Nov-2001 15:30 1k
patched-linux-2.4.15..> 22-Nov-2001 22:18 22.6M
patched-linux-2.4.15..> 22-Nov-2001 22:18 1k
```

```
patched-linux-2.4.15..> 22-Nov-2001 22:18 28.0M
patched-linux-2.4.15..> 22-Nov-2001 22:18 1k
```

Hogyan mondhatnánk meg melyik fájl melyik, anélkül, hogy az egérrel mindegyik hivatkozás felett végig kellene mozogni? A megoldás: állítsuk be olyan módon *httpd.conf* állományunk `IndexOptions` sorát, hogy tartalmazza a `NameWidth` kulcsszót:

```
IndexOptions FancyIndexing NameWidth=*
```

Alapértelmezés szerint az Apache csak a `FancyIndexing` kulcsszót tartalmazza. Adjuk ki az `apachectl restart` parancsot, majd töltsük újra az oldalt, hogy lássuk az eredményt:

```
Index of /~rob/stuff/kernel

Parent Directory 03-Sep-2002 00:33 - Description
patched-linux-2.4.12.tar.bz2 11-Oct-2001 00:59 22.0M
patched-linux-2.4.12.tar.bz2.sign
➥ 11-Oct-2001 00:59 1k
patched-linux-2.4.12.tar.gz 11-Oct-2001 00:59 27.1M
patched-linux-2.4.12.tar.gz.sign
➥ 11-Oct-2001 00:59 1k
patched-linux-2.4.13.tar.bz2
➥ 23-Oct-2001 22:28 22.0M
patched-linux-2.4.13.tar.bz2.sign
➥ 23-Oct-2001 22:28 1k
patched-linux-2.4.13.tar.gz 23-Oct-2001 22:28 27.2M
patched-linux-2.4.13.tar.gz.sign
➥ 23-Oct-2001 22:28 1k
patched-linux-2.4.14.tar.bz2
➥ 05-Nov-2001 15:30 22.1M
patched-linux-2.4.14.tar.bz2.sign
➥ 05-Nov-2001 15:30 1k
patched-linux-2.4.14.tar.gz 05-Nov-2001 15:30 27.4M
patched-linux-2.4.14.tar.gz.sign
➥ 05-Nov-2001 15:30 1k
patched-linux-2.4.15.tar.bz2
➥ 22-Nov-2001 22:18 22.6M
patched-linux-2.4.15.tar.bz2.sign
➥ 22-Nov-2001 22:18 1k
```

```
patched-linux-2.4.15.tar.gz 22-Nov-2001 22:18 28.0M
patched-linux-2.4.15.tar.gz.sign
➥ 22-Nov-2001 22:18 1k
```

Máris sokkal jobb. Telepítsük ezt a szolgáltatást minden Apache rendszerünkre alapértelmezés szerint, és meglátjuk, felhasználóink hálásak lesznek érte.

## #89 Gyors beállításváltás IfDefine használatával

### Változtassuk meg az Apache beállításait a httpd.conf módosítása nélkül

Az Apache lehetővé teszi, hogy beállításait az `IfDefine` és `IfModule` kulcsszavakkal gyorsan és könnyedén változtassuk meg. Az `IfDefine` célja, hogy olyan beállításblokkokat adhassunk meg, amelyek csak az adott parancssoros kapcsolók jelenléte esetén futnak le:

```
# /usr/local/apache/bin/httpd -DSSL
```

A fenti, gyakran előforduló példa egy SSL nevű jelet hoz létre (a `-D` kapcsoló segítségével). Ezzel engedélyezhetjük a következő alakban megadott beállításrészletet:

```
<IfDefine SSL>
# anything in here will be enabled if -DSSL
# has been passed on the command line.
</IfDefine>
```

Ehhez hasonlóan, az `IfModule` kulcsszó azt ellenőrzi, hogy a nevezett modul be lett-e már töltve az Apache beállításfájlban vagy sem. Például, ha a *mod_userdir.c* be van töltve, akkor a következő beállításrészlet végrehajtódik, ellenkező esetben minden további nélkül kimarad.

```
<IfModule mod_userdir.c>
UserDir public_html
</IfModule>
```

Ahogy az Apache növekedett, úgy lett a beállításfájl (*httpd.conf*) is egyre modulárisabb, idővel lehetővé tette, hogy a különböző modulokkal kísérletezve ne kelljen függőségi gondok miatt bekövetkező leállástól tartanunk.

A fenti példák alapján egyszerűen parancssorból is könnyedén megváltoztathatjuk az Apache működését. Tételezzük fel, hogy minden pénteken hírlevelet adunk ki tízezer embernek. Tapasztalatból tudjuk, hogy kiszolgálónknak hétvégeken kell kiállnia a legkomolyabb igénybevételt. Nézzünk egy `IfDefine` kulcsszót használó Apache beállításrészletet:

```
<IfDefine !WEEKEND>
MinSpareServers 1
MaxSpareServers 5
StartServers 1
MaxClients 150
</IfDefine>

<IfDefine WEEKEND>
MinSpareServers 5
MaxSpareServers 15
StartServers 5
MaxClients 300
</IfDefine>
```

Amennyiben a kiszolgálót a szokásos módon indítjuk el a fenti beállításokkal:

```
# /usr/local/apache/bin/httpd
```

a kiszolgálónk indítását és üzemeltetését az alapértelmezett Apache beállításoknak megfelelően fogjuk végezni. Azonban mielőtt pénteken hazamennénk, leállítjuk a démont, majd rögtön újraindítjuk:

```
# /usr/sbin/httpd -DWEEKEND
```

Az újraindítást követően a hétvégi forgalmat hatékonyabban kiszolgáló beállítások hajtódnak végre.

Lássunk egy példát az `IfModule` használatára is:

```
<IfModule libperl.so>
PerlModule Apache::Registry
<Location /usr/local/httpd/cgi-bin>
```

```
SetHandler perl-script
PerlHandler Apache::Registry
Options +ExecCGI
</Location>
</IfModule>
```

A fentiek szerint, ha a `mod_perl` modul betöltődött, akkor az összes (remélhetőleg jól megírt) CGI parancsfájlunkból elérhető lesz a `mod_perl` szolgáltatás. Ez akár az előbbi felállás mellett is hasznos lehet, hiszen a `mod_perl` bekapcsolásával parancsfájljaink gyorsabban válaszolnak majd a megnövekedtet forgalomra. Amennyiben munkanapokon nem szeretnénk feleslegesen futtatni a `mod_perl`-t (esetleg a hibakeresést akarjuk megkönnyíteni), egyszerűen csak megjegyzésbe teszzük a `mod_perl`-t betöltő `LoadModule` és `AddModule` sorokat, és máris minden visszazökken a régi kerékvágásba.

A fenti elképzelések alapján készíthetünk fejlesztőknek szánt különleges beállításokat, amit a `-DFEJLESZTO` kapcsolóval lehet majd meghívni, vagy akár egy `-DVIRTUAL_HOST` lehetőséget is létrehozhatunk, amelyet bármikor eltávolíthatunk, ha az ügyfeleink nem fizetnek:

```
<IfDefine VIRTUAL_HOSTS>
# Ide kerülnek a különböző <VirtualHosts> csoportok
# A -DVIRTUAL_HOSTS kapcsoló (amellyel például azt
# jelezzük, hogy az
# ügyfelünk fizetett) nélkül a virtuális
# kiszolgálók
# nem indulnak el.
</IfDefine>

<IfDefine FEJLESZTO>
# Amikor a -DFEJLESZTO kapcsolót használjuk,
# a szokásos üzemeltetés során kihagyott
# modulok is betöltődnek.
LoadModule proxy_module libexec/httpd/libproxy.so
LoadModule expires_module
libexec/httpd/mod_expires.so
LoadModule usertrack_module
libexec/httpd/mod_usertrack.so
AddModule mod_proxy.c
```

```
AddModule mod_expires.c
AddModule mod_usertrack.c
</IfDefine>
```

Amennyiben az `apachectl` segítségével szoktuk elindítani és leállítani a kiszolgálót, érdemes kicsit módosítani, hogy a kedvenc beállításainkat használja (végtére is egy parancsfájlról van szó). Keressük ki a következő sort:

```
HTTPD=/usr/local/apache/bin/httpd
```

és változtassuk meg valami ilyesmire:

```
HTTPD="/usr/local/apache/bin/httpd -DSSL
➥ -DFEJLESZTO -DVIRTUAL_HOSTS"
```

Ettől kezdve `-D`-s kulcsszavaink minden `apachectl` futtatásakor szabályszerűen adódnak át.

## #90 Egyszerű, hivatkozás alapú hirdetéskövetés

### Készítsünk hirdetéseinkhez egyszerű felhasználó követő rendszert a QUERY_STRING alapján

Mikor magazinokban, újságokban és weblapokon hirdetünk, gyakran érezzük úgy, hogy milyen jó lenne tudni, melyik hirdetés mennyi forgalmat hozott, hiszen így legközelebb célirányosabban költhetjük el pénzünket. Tegyük fel, hogy éppen nincs időnk semmilyen bonyolult dologra, így valami egyszerű, kiskaliberű megoldást keresünk, továbbá valami olyan eszközt, ami segítene elemezni az eredményt.

Szerencsére az Apache örömmel naplózza a külvilág számára szolgáltatott weboldalaknak átadott valamennyi környezeti változót. Így aztán a mi dolgunk sem lesz túl bonyolult: adjunk át hirdetésenként különböző `QUERY_STRING` értéket webhelyünk főoldalának.
A `QUERY_STRING` semmit sem csinál, ha csak nem akarjuk közvetlenül felhasználni (SSI, PHP, CGI vagy hasonló eszköz segítségével), de az Apache ettől még naplózza az eseményt az *acces.log* fájlban.

Tegyük fel, hogy hirdetünk a Magyar Hírlapban. A hirdetésbe mostantól nem csupán azt írjuk, hogy „Látogassa meg honlapunkat a http://www.ErdekesCeg.hu címen", hanem a címet úgy adjuk meg, hogy mindjárt az olvasótábort is jelezze:

```
http://www.ErdekesCeg.hu/?mh
```

Amikor azután valaki begépeli a példában olvasható címet, az Apache továbbra is a megszokott főoldalt fogja felhozni, azonban közben feltűnés nélkül feljegyzi azt is, hogy átadásra került az mh QUERY_STRING érték. Ezt követően tetszőleges naplóvizsgáló programmal (például az Analoggal vagy az alább olvasható egyszerű programocskával) megnézhetjük, pontosan hány ember érkezett honlapunkra a Magyar Hírlapban feladott hirdetésünk alapján.

Mivel QUERY_STRING-et használunk, kiszolgálóoldali megoldásokkal is kezelhetjük őket, például PHP-val, de bármely más kiszolgálóoldali nyelv megfelelő lehet. Az alább olvasható HTML-oldal, például a begépelt webcím alapján különböző üdvözlő szövegeket jelenít meg:

```
<html>
<head>
<title>Apache trükk #90 - Egyszerű hivatkozás alapú
➥ hirdetéskövetés</title>
</head>
<body>
<!--#if expr="\ "$QUERY_STRING\ " = \ "mh\ "" -->
<h1>Üdvözöljük, kedves Magyar Hírlap olvasó!</h1>
<!--#elif expr="\ "$QUERY_STRING\ "
➥ = \ "xml\ "" -->
<h1>Üdvözöljük, kedevs index.hu olvasó!</h1>
<!--#else -->
<h1>Üdvözöljük honlapunkon!</h1>
<!--#endif -->
</body>
</html>
```

A fenti példa különböző üzeneteket jelenít meg aszerint, hogy a felhasználó a Magyar Hírlapban vagy az Index.hu-n megjelent hirdetéseknek megfelelő QUERY_STRING-et gépelte-e be. Amennyiben a fel-

használónk egyiket sem gépeli be, akkor az általános üdvözlőszöveg jelenik meg. Akadnak honlapok, amelyek nemcsak a szöveget, de a színeiket, cégemblémájukat és belső hirdetéseiket is változtatgatják.

Nagyon figyeljünk rá, hogy milyen QUERY_STRING értéket választunk. Hogyha hosszú, nehezen megjegyezhető, netán rémisztő, a felhasználó könnyen elgépelheti a címet (így hibás statisztikáink lesznek) vagy a QUERY_STRING értékét egyáltalán be sem írja. Legrosszabb esetben még a látogatástól is elriasztjuk. A következő példák nem túl jó választások:

```
# Ez túl hosszú.
http://www.gamegrene.com/?newyorktimes-04-21

# Az ilyet nagyon nehéz megjegyezni és begépelni.
http://www.gamegrene.com/?04xlmfo3d

# Ez pedig gyanakvóvá teszi az embereket.
http://www.gamegrene.com/?track=nyt
```

Amennyiben a hálózaton hirdetünk, jól használhatjuk a horgony tagokban elhelyezett kérelemszövegeket (ráadásul ezek akármilyen hosszúak is lehetnek, hiszen a felhasználó szempontjából csak egy kattintás az egész). Lekéréseink naplóvizsgálatához a következő Perl-parancsfájlt javasoljuk:

### Lista: referral-report.pl

```
#!/usr/bin/perl -w
use strict; my %ads;

# define each of your QUERY_STRINGS,
# and the matching "real name" here.
# this script will only look for
# QUERY_STRINGS that are letters and
# numbers only (no dashes, etc.)
$ads{ "xml"}   = "XML.com";
$ads{ "nyt"}   = "New York Times";
$ads{ "ora"}   = "O'Reilly and Associates";

# your Apache access log location.
my $logfile = "/var/log/httpd/access_log";
```

```
# define some counters.
my %ads_count; my $total;

# open the logfile.
open(LOG, "<$logfile") or die $!;

# and loop through each line.
while (<LOG>) {

# skip over lines we're not interested in.
next unless /GET/; next unless /\ ?(\ w+)\ s/;

# save the query_string.
my $query_string = $1;

# move on if not defined, else increment.
next unless exists $ads{ $query_string} ;
$total++; $ads_count{ $query_string} ++;

}

# and print out the data.
print "There were a total of $total
➥ ad referrals.\ n";
foreach ( sort keys %ads_count ) {
print "$ads{ $_}  had $ads_count{ $_}  ad
referrals.\ n";
}

# close logfile and
# exit the program.
close(LOG); exit;
```

A parancsfájl eredménye körülbelül így fog festeni:

```
There were a total of 17 ad referrals.
XML.com had 6 ad referrals.
New York Times had 7 ad referrals.
O'Reilly and Associates had 4 ad referrals.
```

Az egyszerűség kedvéért ez a parancsfájl nem figyeli, hogy azonos IP-címről több hivatkozás érkezett-e, és nem készít százalékos kimutatást sem az összes beérkezett hivatkozás száma alapján. Egy igazi Perl-programozónak bármelyik feladat gyerekjáték.

# #91 Utánozzuk Apache-val az FTP-kiszolgálókat

## Készítsünk többszintű anonim elérést Apache alatt

Tegyük fel, hogy weboldalunk egy részét FTP-szerűen szeretnénk kialakítani: néhány felhasználónak legyen külön letöltési azonosítója, míg a többiek anonim letöltési jogosultságot kapnak, megint másoknak pedig csak bizonyos típusú fájlok letöltését szeretnénk engedélyezni. Mivel nemigen bízunk benne, hogy az ügyfeleink képesek lesznek letölteni és feltelepíteni valamilyen FTP-programot, a szokásos böngésző környezetben próbálunk kialakítani hasonló megoldást.

Az alapértelmezett összeállításban is megtalálható, gyakran mégis méltatlanul elfelejtett Apache modul felhasználásával honlapunk egyes részeit azonosított anonim bejelentkezéssel érhetjük el, illetve tartalékként megtarthatjuk a `mod_auth` megszokott azonosítási módszereit. Ez a módszer jól jöhet akkor, ha például rengeteg dokumentumunk van, és nem szeretnénk, hogy a keresőszolgáltatások feljegyezzék őket, vagy ha több felhasználói szintet szeretnénk létrehozni. Mindezt tisztán a böngészővel, egy sor programozás nélkül is megtehetjük.

Az első lépés az `anon_auth_module` engedélyezése, amely alapértelmezés szerint megjegyzésben áll az Apache beállításfájljában. Keressük ki a következő sort (terjesztésünktől függően kis mértékben másmilyen is lehet) és töröljük elejéről a # (kettős kereszt) jelet:

```
#LoadModule anon_auth_module
libexec/httpd/mod_auth_anon.so
#AddModule mod_auth_anon.c
```

Miután a fenti sorokat kiszedtük a megjegyzések közül, elkezdhetjük begépelni a megfelelő utasításokat a *httpd.conf* fájlba vagy a védett könyvtár *.htaccess* állományába.

A továbbiakban feltételezzük, hogy *.htaccess* fájlokat alkalmazunk. Az alábbi részletet másoljuk *.htaccess* állományunkba, majd mentsük a védendő könyvtárba:

```
AuthName "anonymous/levélcm"
AuthType Basic
Require valid-user

Anonymous orko bender
Anonymous_Authoritative on
```

Az első három utasítást biztosan felismerjük, hiszen ugyanezeket használjuk, amikor szokásos módon `mod_auth`-tal védjük a könyvtárat. Az `Anonymous` utasítás adja meg, hogy mely felhasználóneveket kell anonymous felhasználónak tekinteni (jelen esetben az orko és bender neveket).

Az `Anonymous_Authoritative` kulcsszó vezérli, hogy a jelszót és felhasználói nevet át akarjuk-e adni valamilyen más azonosító folyamatnak. Ha az `on` beállítást választjuk, az anonimitás lesz a mérvadó, függetlenül attól, hogy a felhasználó orko vagy bender néven lépett be, vagy egyáltalán be se engedtük.

Ellenben ha `off`-ra állítjuk, a `mod_auth` minden képességét felhasználhatjuk, azonosíthatunk jelszavak alapján, lehetnek csoportjaink és így tovább. Vessünk egy pillantást az alábbi beállításokra. Ha a felhasználó nem heenieként vagy retrogirl-ként jelentkezett be, a felhasználói név az `AuthUserFile`-hoz kerül, és az ott leírtak szerint lesz elbírálva. Amennyiben ott sem létezik ilyen bejegyzés, a felhasználót nem engedjük be.

```
AuthName "anonymous/levélcím"
AuthUserFile /Library/WebServer/.htpasswd
AuthType Basic
Require valid-user

Anonymous heenie retrogirl
Anonymous_Authoritative off
```

Mint általában, itt is maradhatunk az egyszerűség mellett, esetleg végszükség esetén meg is bonyolíthatjuk a dolgokat. A következő beállítás minden felhasználónak engedélyezi a könyvtárak böngészését (azaz láthatják, hogy mi választék). Az `AuthUserFile` listáján szereplő valamennyi felhasználó elérheti a *.jpg* állományokat, de ugyan-

ezt megteheti bármely anonymous felhasználó is, amennyiben a mrs_decepticon vagy a spiderj néven jelentkezik be, feltéve, hogy írásmódját tekintve helyes elektronikus levélcímet adott meg (azaz van benne @ és . karakter, ugyanis az Apache nem ellenőrzi a kiszolgáló meglétét). Végül *.mp3* állományokat kizárólag az eustace nevű felhasználó tölthet le:

```
AuthType basic
AuthName "anonymous/levélcím"
AuthUserFile /Library/WebServer/.htpasswd

Anonymous mrs_decepticon spiderj
Anonymous_Authoritative off
Anonymous_VerifyEmail on

<Files *.jpg>
Require valid-user
</Files>

<Files *.mp3>
Require user eustace
<Files>
```

Még ennél is továbbmehetünk, beállíthatunk IP-cím vagy gépnév-alapú tiltást, környezeti változókat, és így tovább. Csak el ne felejtsük megjegyzésekkel ellátni művünket, utólag ugyanis elég nehéz lesz visszafejteni tulajdonképpen kinek mihez is van jogosultsága.

Az FTP egyéb képességeit, például a feltöltést sajnos nem lehet lemásolni. Az Apache-csal érkező modulok közül egyik sem képes ilyesmire. Ha mégis ilyesmit szeretnénk, a fájlfeltöltést támogató számos CGI fájl egyikét kell használnunk, vagy magunknak kell kifejlesztenünk egy saját változatot.

# #92 Tömörítsük és forgassuk az Apache naplóit

## Egy apró Perl-parancsfájl segítségével tömöríthetjük összes Apache naplónkat, akkor is, ha többet tartunk belőle a rendszerünkön

A következő hasznos kis parancsfájl önműködően forgatja és tömöríti az összes Apache naplót helyettünk. Elolvassa a *httpd.conf*-ot, végignézi az összes csatolt beállításfájlt is, és megjegyzi az összes *Log* bejegyzéshez tartozó naplófájlt. Valamennyi fájlt átnevezi az eredeti név plusz dátum alakúra, újraindítja az Apache-ot, majd *gzip*-pel tömöríti az összes dátumozott naplófájlt. Ezenkívül a tömörített naplókat a `$gid` változóban beállított csoport tulajdonába helyezi, végül írási vagy olvasási jogosultságokat ad a megadott csoportnak. Ezáltal valamilyen más (nem rendszergazdai jogosultságú) folyamat utólag könnyedén feldolgozhatja a tömörített naplókat.

Jegyezzük be a `cron` táblába és futtassuk esténként (vagy hetente, a forgalomtól függően), hogy rendszeres tömörített naplókészletet gyűjthessünk össze, miközben élő naplófájljaink továbbra is ésszerű méretűek maradnak.

### Lista: logflume.pl

```
#!/usr/bin/perl -w
#
# logflume.pl
#
# Roll over and compress Apache log files,
  ➥ following Includes within the
# httpd.conf (and all other configuration files).
#
#
use strict;
$|++;

my $server_root = "/usr/local/apache";
my $conf = "$server_root/conf/httpd.conf";
my $gid = "wwwadmin";

my (%logs, %included, @files, @gzip);

my $date = `date +%Y%m%d`; chomp $date;
```

```
push @files, $conf;

for $conf (@files) {
open(CONF, "<$conf") ||
➥ die "Cannot open config file $conf: $!\ n";

while (<CONF>) {
chomp;
next if /^(\ s+)?#/;

if (/(Transfer|Custom|Error)Log\ s+(\ S+)(\ s+)/i)
{
$logs{ $2} ++;

} elsif (/^(ResourceConfig|Include)\ s+(\ S+)/i) {
if(!$included{ $2} ) {
push @files, $2;
$included{ $2} ++;
}
}
}

close CONF;
}

for my $logfile (sort keys %logs) {
$logfile = "$server_root/$logfile" unless ($logfile
=~ m|^/|);
rename($logfile, "$logfile.$date");
push(@gzip, "$logfile.$date");
}

system("$server_root/bin/apachectl restart");

for my $logfile (@gzip) {
system("gzip $logfile");
system("chgrp $gid $logfile.gz");
system("chmod 664 $logfile.gz");
}
```

# #93 SSL tanúsítvány készítése és a tanúsítvány-aláírási kérelem

## Készítsünk SSL kulcsot, CSR-t és tanúsítványt az Apache-hoz

Amennyiben használni szeretnénk a `mod_ssl` vagy az `Apache-ssl` csomagokat, szükségünk lesz valamilyen hitelesítés-szolgáltató (Certificate Authority) által aláírt tanúsítványra. Példánkban a *http://propaganda.discordia.eris/* kiszolgálón fogunk ilyen tanúsítványt igényelni. Az OpenSSL segítségével a következőképpen juthatunk új kulcshoz:

```
hagbard@fnord:~/certs$ openssl genrsa 512/1024 \
  > propaganda.discordia.eris.key
warning, not much extra random data,
➥ consider using the -rand option
Generating RSA private key, 512 bit long modulus
..++++++++++++
...++++++++++++
e is 65537 (0x10001)
```

Így csak a titkos kulcshoz jutottunk hozzá, és nem a tanúsítványhoz. Amennyiben kulcsunkat jelszóval is szeretnénk védeni, használjuk a `-des3` kapcsolót:

```
hagbard@fnord:~/certs$ openssl genrsa -des3 512/1024
➥    > propaganda.discordia.eris.key
warning, not much extra random data,
➥ consider using the -rand option
Generating RSA private key, 512 bit long modulus
.......++++++++++++
.....++++++++++++
e is 65537 (0x10001)
Enter PEM pass phrase:
```

Azonban ne feledjük: a megadott jelszót az Apache minden egyes újraindításakor meg kell majd adnunk, ami elég kényelmetlen lehet, főként, ha gyakran végzünk karbantartást (például HTTP-naplókat forgatunk). Mérlegeljük, hogy a kényelmetlenség vagy az esetleges rosszindulatú kulcstolvaj által okozott kár esik-e nagyobb súllyal a latba. Ha elveszítjük a jelszavunkat, visszaállítása gyakorlatilag lehetetlen, úgyhogy jól vigyázzunk rá!

Következő lépésként létre kell hoznunk a tanúsítvány-aláírási kérelmet, amelyet majd aláírásra továbbküldhetünk egy megbízható tanúsítvány-szolgáltatónak (ilyen például a Thawte/VeriSign). A vastag betűvel szedett részeket be kell gépelnünk, természetesen a megfelelő helyekre a saját adatainkat kell behelyettesítenünk:

```
hagbard@fnord:~/certs$ openssl req -new -key
propaganda.discordia.eris.key \
   > propaganda.discordia.eris.csr


Using configuration from /usr/local/ssl/openssl.cnf
You are about to be asked to enter information
that will be incorporated
into your certificate request.
What you are about to enter is what is called
➥ a Distinguished Name or a DN.
There are quite a few fields but you can
➥ leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:US
State or Province Name (full name)
➥ [Some-State]:Texas
Locality Name (eg, city) []:Mad Dog
Organization Name (eg, company)
➥ [Internet Widgits Pty Ltd]:Discordia, Inc.
Organizational Unit Name (eg, section)
[]:Operations
Common Name (eg, YOUR name)
[]:propaganda.discordia.eris
Email Address []:norton@discordia.eris

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Ezek után már készen állunk a tanúsítvány létrehozására. Amennyiben valamilyen jól ismert tanúsítvány-szolgáltatót szeretnénk használ-

ni, el kell hozzá küldenünk aláírásra a *.csr* állományunkat. Addig is írjuk alá magunk a tanúsítványunkat, és addig ezt használjuk, amíg a CSR feldolgozására várunk:

```
hagbard@fnord:~/certs$ openssl req -x509 \
  -key propaganda.discordia.eris.key \
  -in propaganda.discordia.eris.csr \
  > propaganda.discordia.eris.crt

Using configuration from /usr/local/ssl/openssl.cnf
hagbard@fnord:~/certs$
```

Amennyiben saját hitelesítés-szolgáltatót állítunk fel, *saját kulcsunkat* használjuk tanúsítványunk aláírására.

### Kapcsolódó anyagok

- [#94] Készítsünk saját hitelesítés-szolgáltatót
- OpenSSL honlap: `http://www.openssl.org/`

## #94 Készítsünk saját hitelesítés-szolgáltatót

### Legyen saját hitelesítés-szolgáltatónk, és írjuk alá saját (vagy mások) SSL tanúsítványait

Mint ismeretes, léteznek olyan hitelesítés-szolgáltatók (például a Thawte/VeriSign), amelyek hivatalos, megbízható harmadik félként adnak ki tanúsítványokat. Üzleti tevékenységük lényege, hogy érzékeny adatokkal (például jelszavakkal, számlaszámokkal) dolgozó weboldalak által használt SSL tanúsítványokat írnak alá. Amennyiben a webhely SSL tanúsítványát egy megbízható szolgáltató ellenjegyezte, akkor vélhetően ellenőrizni tudjuk a tanúsítványt szolgáltató gép hitelesítő adatait is. Ahhoz, hogy egy ismert hitelesítés-szolgáltató „áldását adja" tanúsítványunkra, nemcsak azt kell minden kétséget kizáróan bizonyítanunk, hogy azok vagyunk, akiknek állítjuk magunkat, hanem, hogy jogosultak is vagyunk a tanúsítvány adott típusú felhasználására. Én például valószínűleg bizonyítani tudom a hitelesítés-szolgáltatónak, hogy tényleg én vagyok Rob Flickenger, de valószí-

nűleg nem fognak nekem aláírni egy Microsoft Corporation névre kiállított tanúsítványt, hiszen nincs jogom felhasználni ezt a nevet. Nos, valószínűleg tényleg nem teszik meg. Többé már nem.

Az OpenSSL rendszerben mindent megtalálunk, ami csak a saját hitelesítés-szolgáltatónk felállításához szükséges. A *CA.pl* eszköz pedig jelentősen leegyszerűsíti a folyamatot.

A példában csak a vastagon szedett részeket kell begépelnünk, illetve szükség esetén meg kell adnunk a jelszavakat (ezek nem jelennek meg a képernyőn). Saját hitelesítés-szolgáltatónkat a következőképpen készíthetjük el:

```
hagbard@fnord:~/certs$ /usr/local/ssl/misc/CA.pl
➥ -newca
CA certificate filename (or enter to create)

Making CA certificate ...
Using configuration from /usr/local/ssl/openssl.cnf
Generating a 1024 bit RSA private key
..............++++++
.....................................++++++
writing new private key to
➥ './demoCA/private/cakey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter
➥ information that will be incorporated
into your certificate request.
What you are about to enter is what is
➥ called a Distinguished Name or a DN.
There are quite a few fields but you can
➥ leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name)
➥ [Some-State]:California
Locality Name (eg, city) []:Sebastopol
```

```
Organization Name (eg, company) [Internet
➥ Widgits Pty Ltd]:Illuminatus Enterprises, Ltd
Organizational Unit Name (eg, section)
➥ []:Administration
Common Name (eg, YOUR name) []:Hagbard Celine
Email Address []:hagbardceline1723@yahoo.com
```

Gratulálunk! Mostantól saját hitelesítés-szolgáltatóval rendelkezünk. Nézzünk is egy kicsit körül:

```
hagbard@fnord:~/certs$ ls
demoCA/
hagbard@fnord:~/certs$ cd demoCA/
hagbard@fnord:~/certs/demoCA$ ls -l
total 24
-rw-r--r-- 1 rob users 1407 Sep 8 14:12 cacert.pem
drwxr-xr-x 2 rob users 4096 Sep 8 14:12 certs/
drwxr-xr-x 2 rob users 4096 Sep 8 14:12 crl/
-rw-r--r-- 1 rob users 0 Sep 8 14:12 index.txt
drwxr-xr-x 2 rob users 4096 Sep 8 14:12 newcerts/
drwxr-xr-x 2 rob users 4096 Sep 8 14:12 private/
-rw-r--r-- 1 rob users 3 Sep 8 14:12 serial
```

Új hitelesítés-szolgáltatónk nyilvános kulcsát a *cacert.pem* állományban találjuk, a titkos kulcs pedig a *private/cakey.pem* állományban kapott helyet. Ezt a titkos kulcsot használhatjuk az SSL tanúsítványok aláírására.

Ahhoz, hogy SSL tanúsítványokat írhassunk alá hitelesítés-szolgáltatónkban, először is készítenünk kell egy új tanúsítványt, amit a webkiszolgáló (például az Apache) felhasználhat. Készítsünk egy saját kulcsot és tanúsítvány-aláírási kérelmet a [#93] „SSL tanúsítvány készítése és a tanúsítvány-aláírási kérelem" trükk útmutatása szerint. Az új kérelmet mindjárt alá is írhatjuk saját hitelesítés-szolgáltatónk kulcsával:

```
hagbard@fnord:~/certs$ openssl ca -policy
➥ policy_anything \
  -out propaganda.discordia.eris.crt \
  -infiles propaganda.discordia.eris.csr
```

```
Using configuration from /usr/local/ssl/openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName :PRINTABLE:'US'
stateOrProvinceName :PRINTABLE:'Texas'
localityName :PRINTABLE:'Mad Dog'
organizationName :PRINTABLE:'Discordia, Inc.'
organizationalUnitName:PRINTABLE:'Operations'
commonName :PRINTABLE:'propaganda.discordia.eris'
emailAddress :IA5STRING:'hail@discordia.eris'
Certificate is to be certified until Sep 8 22:49:26
2003 GMT (365 days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified,
➥ commit? [y/n]y
Write out database with 1 new entries

Data Base Updated
```

A létrejött *.crt* és *.key* állományokat használjuk fel az Apache `mod_ssl`-el (vagy az *Apache-SSL*) rendszerhez. Telepítsük őket a megszokott módon (valószínűleg a következő sorokkal):

```
SSLCertificateFile \
/usr/local/apache/conf/ssl.crt/
➥ propaganda.discordia.eris.crt
SSLCertificateKeyFile \
/usr/local/apache/conf/ssl.key/
➥ propaganda.discordia.eris.key
```

Ez igazán izgalmas volt, de mi lesz, ha egy ügyfél mégis csatlakozik a *http://propaganda.discordia.eris/* kiszolgálóhoz? Nem fog a böngésző hibát jelezni, mert nem ismerte fel a SSL tanúsítványt aláíró hitelesítés-szolgáltatót? Dehogyisnem, amennyiben nem telepítettük a hitelesítés-szolgáltató nyilvános kulcsát már jó előre a böngészőnkbe. Nézzük meg a következő módosítást, ha ilyesmit szeretnénk.

### Kapcsolódó anyagok

- `man CA.pl`
- Az OpenSSL honlap a `http://www.openssl.org/` címen érhető el.
- `http://www.cert.org/advisories/CA-2001-04.html`

Cáfolat: teljesen őszintén mondom, hogy az égvilágon semmi közöm nincsen a hamis Microsoft Corporation tanúsítványügyhöz. Viszont ez nagyon szépen példázza a hálózati élet egyik alaptételét: nehéz megmondani, *kiben* bízhatunk.

## #95 Hitelesítés-szolgáltatónk terjesztése

### A gyönyörű új hitelesítés-szolgáltatónk tanúsítványának telepítése csak egy kattintás a böngészőnkben

A böngészők két formátumban a *pem* és a *der* szabvány szerint hajlandók elfogadni tanúsítványokat. A Netscape korai változati csak a *pem* formátumot ismerték, de a jelenlegiek már bármelyiket elfogadják. Az Internet Explorer éppen ellenkezőleg fejlődött (a korai változatok csak a *der* formátumot ismerték, a jelenlegi változatai pedig bármelyiket elfogadják). A többi böngésző általában bármelyik változatot elfogadja. A már létező *pem* állományunkat egyetlen OpenSSL paranccsal átalakíthatjuk *der* formátumúvá:

```
hagbard@fnord:~/certs$ openssl x509
➥ -in demoCA/cacert.pem \
  -outform DER -out cacert.der
```

Ezen felül Apache telepítésünk *conf/mime.types* állományába is be kell jegyeznünk a következőket:

```
application/x-x509-ca-cert der pem crt
```

Indítsuk újra az Apache-ot, hogy az új beállítások végrehajtódjanak. Ezek után a *cacert.der* és *demoCA/cacert.pem* állományainkat kiszolgálónkon tetszőleges helyre feltehetjük, az ügyfelek egyetlen kattintással feltelepíthetik őket böngészőjükbe.

Amikor letöltjük az új tanúsítványt, a böngésző egy párbeszédablakban rákérdez, hogy tényleg folytatni akarjuk-e a telepítést. Mindössze annyit kell tennünk, hogy elfogadjuk a tanúsítványt. Mostantól a hitelesítés-szolgáltatónk által aláírt tanúsítványokat kérdés nélkül elfogadja az ügyfél böngészője.

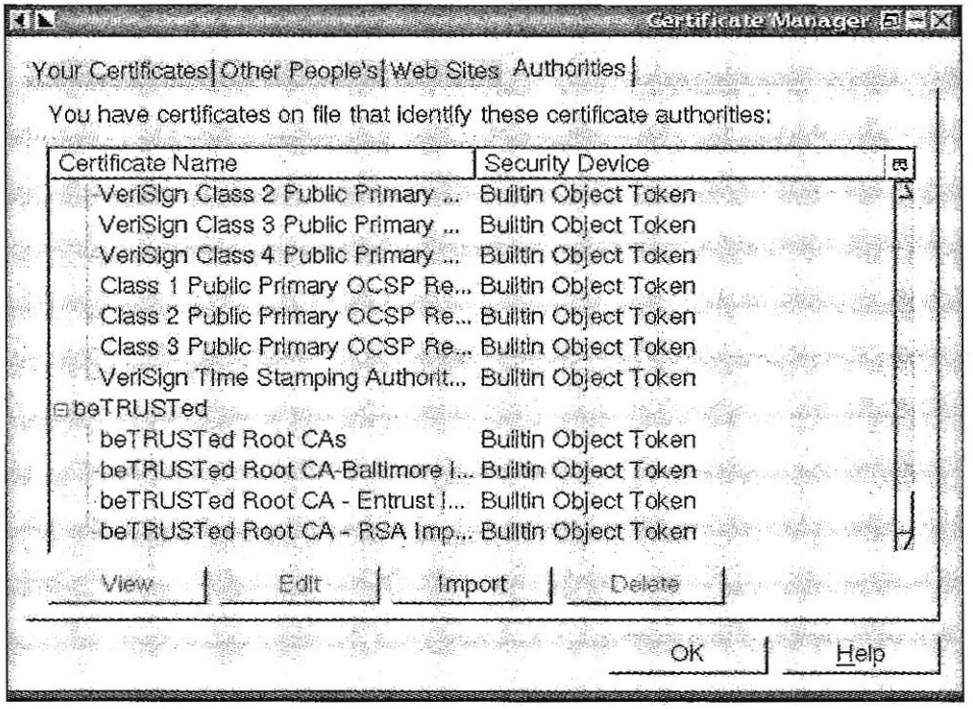
Ne feledjük el azonban, hogy a hitelesítés-szolgáltatókat soha nem szabad könnyelműen kezelni. Csak akkor fogadjunk el böngészőnkkel ilyesmit, ha valóban teljesen megbízunk benne. Egy rosszhiszemű hitelesítésszolgáltató-tulajdonos olyan tanúsítványokat is aláírhat, amelyekben magunktól soha nem bíznánk meg, a böngészőnk viszont nem fog visszakérdezni (hiszen azt mondtuk neki, hogy megbízunk a hitelesítés-szolgáltatóban). Ha SSL-képességekkel felruházott böngészőt használunk, nagyon ügyeljünk rá, kinek adjuk bizalmunkat. Sőt, az sem árt, ha körbeszaglászunk kicsit a hitelesítésszolgáltató-gyorstárban, hogy azt is pontosan lássuk, kikben bízunk meg alapértelmezés szerint.

Például hányan tudják, hogy az AOL/Time Warner saját hitelesítésszolgáltatót tart fenn? És a GTE? Vagy a VISA? A Netscape 7.0 for Linux telepítéssel együtt ezek (többek közt) mind megbízható szolgáltatóként kerülnek a gépünkre, a weboldalak, levelek és alkalmazásletöltés csoportjában egyaránt. Tartsuk szem előtt amennyiben SSL-kezelő honlapon járunk: ha az alapértelmezett hitelesítés-szolgáltatók közül bármelyik is ellenjegyezte a hálózati tartalmat, a böngészőnk kérdés nélkül meg fog bízni benne. A 8.1. ábrán láthatjuk hivatalok tárát.

Amennyiben fontosnak tartjuk a böngészőnk biztonságát (és ennek folyományaként ügyfélgépünk biztonságát), ne restelljük felülvizsgálni hitelesítés-szolgáltatónk kapcsolatait.

## Kapcsolódó anyagok

- Az OpenSSL FAQ címe
  `http://www.openssl.org/support/faq.cgi`
- [#93] SSL tanúsítvány készítése és a tanúsítvány-aláírási kérelem
- [#94] Készítsünk saját hitelesítés-szolgáltatót




8.1. ábra

*Nem árt, ha jól megnézzük pontosan kiben is bízik meg böngészőnk*

# #96 Több webhely kiszolgálása azonos DocumentRoot alól

## A mod_rewrite cseles felhasználásával elérhetjük, hogy több hely osztozzon egyazon DocumentRoot könyvtáron, miközben továbbra is független webhelyeknek látszanak

Néha előfordul, hogy azonos *DocumentRoot* könyvtárat szeretnénk több oldalhoz is felhasználni, amelyek azonban mégis megtarthatják saját egyedi kezdőoldalukat. Például előfordulhat, hogy szeretnénk megosztani a kép vagy *JavaScript* könyvtárunkat, de valamiért nem akarunk különböző álnevek beállításával bajlódni. Avagy a teljes könyvtárszerkezetet meg akarjuk osztani, de nem szeretnénk engedélyezni a `FollowSymlinks` kapcsolót. Bármilyen okból is szeretnénk közös *DocumentRoot* könyvtárat, a `mod_rewrite` lesz a megoldás.

Tegyük fel, hogy mi üzemeltetjük a *www.gyumolcs.yum* nevű honlapot, és almákról, illetve narancsokról szeretnénk tájékoztatást nyújtani. Eddig is nagy sikerünk volt a *http://www.gyumolcs.yum/alma* és a *http://www.gyumolcs.yum/narancs* címekkel, de valahogy sikerült szert tennünk a *www.alma.yum* és *www.narancs.yum* tartománynevekre is. Szeretnénk megőrizni a *www.gyumolcs.yum* alatt használt

könyvtárszerkezetet, hátha valaki az új tartománynevek alatt próbálja meg elérni az adatokat (például a *http://www.alma.yum/rendeles* vagy a *http://www.narancs.yum/ gyik* oldalakat el szeretné érni), ugyanakkor külön egyedi oldalt szeretnénk megjeleníteni amikor a felhasználó a kezdőoldalon nyitja meg (például *http://www.alma.yum/*).

A `mod_rewrite` segítségével ez nem okozhat gondot. Tételezzük fel, hogy a *www.gyumolcs.yum* `VirtualHost` bejegyzést létrehoztuk a következő formában:

```
<VirtualHost *>
ServerName www.gyumolcs.yum

ServerAdmin webmaster@gyumolcs.yum

DocumentRoot /home/www/htdocs
CustomLog /home/www/logs/access_log combined
ErrorLog /home/www/logs/error_log
</VirtualHost>
```

Illesszünk be pár új `VirtualHost` bejegyzést az új tartományneveinknek és mindjárt egészítsük is ki őket néhány `RewriteRule` sorral:

```
<VirtualHost *>
ServerName www.alma.yum

ServerAdmin webmaster@gyumolcs.yum

DocumentRoot /home/www/htdocs
CustomLog /home/www/logs/access_log combined
ErrorLog /home/www/logs/error_log

RewriteEngine On

RewriteRule ^/$ /home/www/htdocs/alma/index.html
RewriteRule ^/index.html$
/home/www/htdocs/apples/index.html
</VirtualHost>
```

```
<VirtualHost *>
ServerName www.narancs.yum

ServerAdmin webmaster@gyumolcs.yum

DocumentRoot /home/www/htdocs
CustomLog /home/www/logs/access_log combined
ErrorLog /home/www/logs/error_log

RewriteEngine On

RewriteRule ^/$ /home/www/htdocs/narancs/index.html
RewriteRule ^/index.html$
/home/www/htdocs/oranges/index.html
</VirtualHost>
```

Mivel belső újraírást alkalmaztunk, az ügyfél böngészőjében látható URL nem fog megváltozni. Az ő szemszögéből nézve a *http://www.narancs.yum/* teljesen másik oldalnak fog tűnni. Ha közvetlenül a *http://www.gyumolcs.yum/narancs/* oldalra lép be, ott is azonos működésű oldalt talál, feltéve, hogy az összes HTML-oldalunkon közvetett hivatkozásokat (relative links) vagy a megfelelő álnév sorokat (Alias lines) használtunk. Amennyiben még ennél is általánosabb megoldást szeretnénk, ami tetszőleges számú *www.*.yum* címre működik, a következő beállításokat próbáljuk ki:

```
<VirtualHost *>
ServerName www.gyumolcs.yum
ServerAlias www.*.yum

ServerAdmin webmaster@gyumolcs.yum

DocumentRoot /home/www/htdocs
CustomLog /home/www/logs/access_log combined
ErrorLog /home/www/logs/error_log


RewriteCond %{ HTTP_HOST}   www.(.*).yum
RewriteCond %{ HTTP_HOST}   !www.gyumolcs.yum
RewriteRule (.*) $1 [E=SITE:%1]
```

```
RewriteCond %{ REQUEST_URI}  ^/index.html$
RewriteCond /home/www/htdocs/%{ ENV:SITE}
/index.html -f
RewriteRule .* /home/www/htdocs/%{ ENV:SITE}
/index.html

RewriteCond %{ REQUEST_URI}  ^/$
RewriteCond /home/www/htdocs/%{ ENV:SITE}
/index.html -f
RewriteRule .* /home/www/htdocs/%{ ENV:SITE}
/index.html
</VirtualHost>
```

Ebben a beállításban kivágjuk a kért tartománynév középső részét (a *www.* és a *.yum* közé eső szakaszt), majd ellenőrizzük, hogy létezike az oldalhoz tartozó *index.html* fájl a *DocumentRoot* ilyen nevű alkönyvtárában. Amennyiben létezik és a kérelem */* vagy */index.html* alakú volt, végrehajtjuk a megfelelő belső újraírást.

A `mod_rewrite`-tal nem könnyű dolgozni. Szerencsére létezik olyan eszköz, amelyik nagyon sokat segíthet az újraírási gondok felderítésében: ez a `RewriteLog` képesség. Ha elvesztettük a fonalat valamelyik `RewriteRule`-értelmezésénél, írjunk be pár ehhez hasonló sort a `VirtualHost` szakaszunkba:

```
RewriteLog /tmp/rewrite.log
RewriteLogLevel 9
```

Ezzel bekapcsoltuk a megadott fájlba irányított teljes körű naplózást. Érdemes a `tail -f /tmp/rewrite.log` vagy a `less -f /tmp/rewrite.log` parancsokkal vizsgálódni, mialatt különböző URL-eket gépelünk be a böngészőnkbe. Természetesen éles kiszolgálókon ilyesmit ne hagyjuk fenn túl sokáig, hiszen így minden újraírás naplózódik (következésképpen az Apache lassabban fog futni).

# #97 Tartalomszolgáltatás lekérdező karakterlánc alapján, mod_rewrite segítségével

## Vezéreljük a tartalomszolgáltatást CGI parancsfájl nélkül, címlekérdező karakterlánc alapján

Néha hasznos lehet, ha a címsor lekérdező karakterlánca alapján (ez az, ami az első ? (kérdőjel) mögött áll) tudjuk vezérelni az Apache által szolgáltatott tartalmat. A lekérdező karakterláncot általában CGI parancsfájlok használják programváltozók írására és olvasására, a `mod_rewrite` azonban külső parancsfájl nélkül is képes felhasználni őket.

Képzeljük el, hogy olyan terjesztőrendszerünk van, amely a nagyobb lélegzetű műveket oldalakra bontja fel. A tartalmat hagyományos esetben egy parancsfájl szolgáltatja, amely feldolgozza az `oldal=` lekérdező karakterláncot és visszaadja a megfelelő számú oldalt. Ha minden egyes oldalt le tudunk menteni a helyi merevlemezre, akkor `RewriteRule` szabályokat is használhatunk a tárolt másolatok megjelenítéséhez, s így nem kell minden találatkor újra lefuttatni a parancsfájlt.

Tételezzük fel, hogy a cikk második oldalát a *http://honlapom.com/hirek/cikk.html?lap=2* címen szolgáltatjuk. Helyezzük el a következő szabályokat:

```
RewriteCond %{ QUERY_STRING}  page=([0123456789]+)
RewriteCond /home/www/htdocs/%{ REQUEST_URI} .%1 -f
RewriteRule .* /home/www/htdocs/%{ REQUEST_URI}
➥ .%1 [L]
```

Amennyiben létezik a */home/www/htdocs/hirek/cikk.html.2* nevű állomány, akkor belső átirányításként fut le, majd kilép, mielőtt a terjesztő parancsfájlhoz egyáltalán eljutna. Ha a fájl még nem létezik, a feldolgozás az utolsó szabályig lefut, és így a megszokott terjesztési módszer szerint készül el (vagyis feltételezhetően egy *news* nevű `ScriptAlias` utasítással megadott parancsfájl segítségével). Ez a módszer bármilyen egész számú oldalra működik.

Ugyanakkor a lekérdező karakterlánc hiánya is bírhat jelentéssel. Tételezzük fel, hogy tartalomszolgáltató parancsfájlunk bizonyos változók alapján módosítja a tartalmat, de ha nem adunk meg semmilyen változót, akkor valamilyen alapértelmezett tartalmat ad vissza (például a cikk első oldalát). Ez is egy olyan eset, ahol jól megtervezett `RewriteRule`-szabályokkal megtakaríthatunk némi erőforrást (vagyis nem pazarolunk annyit), amit egyébként a dinamikus megjelenítésre fordítanánk:

```
RewriteCond %{ QUERY_STRING}   =""
RewriteCond /home/www/static/%{ REQUEST_URI}   -f
RewriteRule .* /home/www/static/%{ REQUEST_URI}
➥[L]
```

Itt most feltételeztük, hogy ha a lekérdező karakterlánc létezik, akkor az mindig dinamikus tartalomra vonatkozik. Amennyiben ez mégsem így lenne, először meg kell néznünk, hogy létezik-e helyi másolat a kért tartalomból (a */home/www/statikus/* könyvtár alatt), és ha igen, akkor azt adjuk vissza. Ha helyi másolat nem létezik, a feldolgozás a következő utasításcsoportra lép rá (amely valószínűleg lefuttatja a dinamikus tartalomkészítőt). Mindaddig, amíg valamilyen külső folyamat gondoskodik róla, hogy a régi tartalmak eltűnjenek a gyorstárból (`cron` programmal, vagy külön e célra írt, webtartalmat figyelő démon segítségével) az egész gyorstárazás a felhasználó szemszögéből nézve láthatatlan lesz. A régi tartalom eltüntetéséhez egyszerűen töröljük a gyorstárazott változatot, így az a következő találatkor már a dinamikusan jön létre.

Lekérdező karakterláncok segítségével tetszés szerinti utasításokat adhatunk az Apache-nak. A `mod_rewrite` hatékonyságának tényleg csak a képzeletünk (és szkif-tudásunk mértéke) szabhat határt.

## Kapcsolódó anyagok

- [#99]
- [#100]
- Az Apache `mod_rewrite` útmutatóját a `http://httpd.apache.org/docs/mod/mod_rewrite.html` címen találjuk.

# #98 Gyorsítsunk az Apache-on mod_proxy-val

## Töltsünk át bonyolult dinamikus lekérdezéseket másik Apache-ba (vagy akár másik kiszolgálóra)

Iszonyatos mennyiségű munka van amögött, hogy a fájlrendszer fájljait az Apache a lehető leggyorsabban csippentse fel és dobja vissza válaszul a beérkező HTTP-kérelemre. Azok a honlapok, amelyek kizárólag fájlrendszeren tárolt állományokban csücsülő tartalmat tudnak nyújtani, sajnos nem tartoznak a legnépszerűbbek közé. Az interaktív tartalom iránti óriási igény hatására aztán sorra megszülettek azok a megoldások, amelyek a végfelhasználónak testreszabható, dinamikusan változó böngészőélményt tudnak nyújtani.

Sajnálatos módon, ahogy az oldal interaktív teljesítménye nő, a teljesítménytől általában már el is búcsúzhatunk. Az interaktív tartalomhoz programozási nyelvre van szükség, a tetszőleges kód értelmezése pedig igen erőforrás-igényes (különösen az állandó fájlok szolgáltatásához képest). Az egyik klasszikus elrettentő példa a Perl CGI, amely az adatbázis-műveleteket és levélküldést a külső *Sendmail* program meghívásával éri el. A statikus állományok kiadási idejével összehasonlítva, egyetlen CGI-művelet is szinte az örökkévalóságig tart (következésképpen sok lekérdezés sokszoros örökkévalóságig, így kiszolgálónk csigalassúsággal fog vánszorogni).

Az Apache `mod_perl` és `mod_php` moduljai (és egy nagy adag programozói gyakorlat) segítségével sokat javulhat a helyzet, mivel nem kell többé minden egyes találatnál külső programot hívogatni. De a dinamikus kérelmek válaszideje még az Apache-ba épített értelmezők alkalmazásával is ritkán közelíti meg a statikus kiszolgálók sebességét. Ennek egyik oka, hogy legtöbb találat teljesítménye csökken a beágyazott nyelvek használata miatt, függetlenül attól, hogy a kérelem valóban dinamikus volt-e vagy sem.

Képzeljünk el egy komolyabb `mod_perl` telepítést. Néhány perces működést követően egyetlen folyamat körülbelül 40–50 MB memóriát fogyaszt, mivel a Perl-modulok a gyors végrehajtás érdekében a memóriában tárolódnak. Most képzeljük el, hogy egyképpontos átlátszó *.gif* képre érkezik kérelem (hiszen ez egy általánosan használt

trükk, amellyel a webtervezők képpontra pontos tetszőleges méretű üres helyet tudnak létrehozni a weboldalon). A legrosszabb esetet feltételezve, már minden `httpd` folyamat foglalt, így az Apache úgy dönt, újat hoz létre. A kiszolgálónak el kell indítania a `mod_perl` mamutot, létrehoz egy gyermeket, amely megeszik 50 MB memóriát és rengeteg feldolgozási időt, s mindezt azért, hogy egy néhány bájt hosszú statikus állományt visszaadjon. Ha tehát sikerülne elérni, hogy a `mod_perl` gigász csak akkor induljon el, amikor tényleg szükség van rá, nyilván minden egyes alkalommal rengeteg időt nyernénk, amikor valaki csak egy képet (vagy bármilyen más, interaktivitást nem igénylő állományt) szeretne elérni.

Képzeljük el, hogy két Apache alkalmazásunk fut egyszerre: az egyikben benne van minden modul, csingilingi, síp, dob, zene, amire csak a dinamikus tartalmunknak szüksége van, és egy másik „kopasz" darab, amelyet a lehető legegyszerűbben alakítottunk ki. Minden kérelmet a pehelysúlyú Apache szolgál ki, a statikus tartalmat a megszokott módon visszaadhatjuk, a dinamikus kéréseket pedig átirányítjuk a dinamikus tartalomszolgáltatóhoz.

Tegyük fel, hogy a lecsupaszított Apache-ot a hagyományos 80-as kapun futtatjuk, az alkalmazáskiszolgálónk pedig a 8088-as kapura kerül. A következő újraíró szabállyal mindent, ami nem kép vagy mentésadat továbbítunk az alkalmazáskiszolgálónak. A következő sorokat helyezzük webhelyünk *VirtualHost* szakaszába (a 80-as kapun futó kiszolgálón):

```
RewriteEngine On

RewriteCond %{ REQUEST_URI}   !.*\
.(jpg|gif|pdf|png|zip|tgz|gz)$
RewriteRule ^/(.*) http://%{ HTTP_HOST}
➥:8088/$1 [P]
```

A `RewriteRule` sor végén álló `[P]` jelöli a proxyműködést. Ennek hatására a pehelysúlyú Apache kapcsolatba lép a dinamikus kiszolgálóval a 8088 kapun keresztül, és úgy tesz, mintha a kérelmet maga az ügyfél küldte volna. Amint megkapja a választ, rögtön tovább is adja az ügyfélnek, mintha maga szolgáltatta volna azt. Az ügyfél szemszö-

géből nézve csak egyetlen webkiszolgáló létezik: egy nagyon gyors, a 80-as kapun. Amíg két Apache telepítésünk *DocumentRoot* könyvtárai megegyeznek, a módszer kitűnően működik.

Ha az alkalmazásunk sütiket (cockies) is használ, az előbbi újraírási szabályok elé írjunk egy `ProxyPassReverse` sort:

```
ProxyPassReverse / http://%{ HTTP_HOST} :8088/
```

Ezzel arra utasítjuk az Apache-ot, hogy a proxy használata közben fordítsa le a HTTP-fejléceket, így a sütik megfelelő módon utazhatnak a kiszolgálóról az ügyfél gépére.

Amennyiben elég alkatrészünk van, a terhelést olyan módon is szétoszthatjuk, hogy az egyik gépet kizárólag proxykiszolgálónak használjuk, a másik pedig az alkalmazáskiszolgálónk lesz. Ez esetben a következő szabállyal próbálkozzunk:

```
ProxyPassReverse /
http://.alkalmazas.kiszolgalonk.cime/

RewriteEngine On

RewriteCond %{ REQUEST_URI}   !.*\
.(jpg|gif|pdf|png|zip|tgz|gz)$
RewriteRule ^/(.*)
➥ http://.alkalmazas.kiszolgalonk.cime/$1 [P]
```

Természetesen, ha fizikailag is több kiszolgálónk van, valamilyen módon össze kell hangolnunk a fájlrendszereket (erről bővebben olvashatunk a [#41] „Fájlrendszerek egyes részeinek összehangolása az rsync paranccsal" trükkben), máskülönben az emberek minden frissítés után oldalütközésekkel fognak találkozni. Amennyiben még két kiszolgáló sem elégséges az összes dinamikus tartalmunk kezelésére, olvassuk el a [#99] „Terhelésmegosztás Apache RewriteMap-pel" trükkben, ahol megmutatjuk hogyan oszthatjuk meg a terhelést tetszőleges számú gép között.

# #99 Terhelésmegosztás Apache RewriteMap-pel

## Növeljük tetszőlegesen webes alkalmazáskiszolgálóink számát a RewriteMap segítségével

Az előző módosításban láthattuk, hogy a `mod_proxy`, valamint a `mod_rewrite` alkalmazásával átlátszó módon tudunk bármely más webkiszolgáló tartalmából adatot szolgáltatni. Ha az `[R]` külső átirányítás helyett a `[P]` jelöléssel proxycélpontot adunk meg, az ügyfél számára úgy fog tűnni, mintha az adatok az eredetileg megcímzett kiszolgálóról érkeznének, miközben az ügyfél böngészőjének URL sora sem változik meg. Mint korábban láthattuk, e módszer egyik felhasználási területe, amikor két gép közt osztjuk meg a terhelést, így azok messze több kérelmet tudnak kiszolgálni, mint amennyire egyetlen gép képes lenne.

Nos, de mi a helyzet, ha nekünk ennél is nagyobb forgalmunk van, és egyetlen alkalmazáskiszolgáló többé már nem elegendő? Olyan módszerre van szükségünk, amely lehetővé teszi, hogy proxykiszolgálónkat az elérhető alkalmazáskiszolgálók készletéből tudjuk kiválasztani, lehetőleg olyan módon, hogy az erősebb kiszolgálókra több találat jusson. Ezt a képességet a `RewriteMap` kulcsszó teszi elérhetővé.

Lássunk egy példát a `RewriteMap` alkalmazására:

```
RewriteMap server
rnd:/usr/local/apache/conf/servers.map
```

A `RewriteMap` kulcsszóval létrehoztunk egy server nevű listát, amely fizikailag a *conf/* könyvtárban található *servers.map* fájlhoz kapcsolódik. Az `rnd:` véletlenszerűen összekeveri a szöveges listát.

A *servers.map* fájl formátuma meglehetősen egyszerű:

```
web  www1 | www2 | www3 | www4 | www5
```

A bal oldalon tetszőleges változónév állhat, amit beállításfájlunkban bárhol felhasználhatunk, így akár a `RewriteRule`-szabályunkban is:

```
RewriteRule ^/(.*) http://${ server:web}
.oreillynet.com/$1 [P]
```

A *servers.map* állomány sorainak jobb oldalán cső- (pipe) karakterekkel elválasztott karaktersorozatok állnak, amelyeket véletlen sorrendben kapunk vissza. A fenti `RewriteRule` esetében a `${server:web}` kifejezés helyére a *servers.map*-ból választott véletlen érték (valamelyik a *www1*, *www2*, *www3*, *www4* vagy *www5* karaktersorozatok közül) kerül. Ennek eredményképpen proxyszabályunk értéke a *http:/www4.oreillynet.com/* (vagy bármely másik *www**kiszolgáló) és az eredeti URI együttese lesz.

A teljesen véletlen listákkal csak az a gond, hogy a benne felsorolt gépekre közel azonos mennyiségű forgalom jut az idők során. Viszont ha a www1 gép négyprocesszoros 2,4 GHz-es Xeon, 8 GB memóriával, miközben a www2 csak egy 486 SX/33 4 MB memóriával, akkor igen valószínű, hogy a www1 gépünkre szeretnénk valamivel több terhelést küldeni. Ehhez semmi mást nem kell tennünk, minthogy az erősebb gépeket többször soroljuk fel a *servers.map* listában:

```
web www1|www1|www1|www1|www2|www3|www3|www4|www5|www5
```

Ebben a példában a www1 tízből négyszer kap találatot, a www2 tízből csak egyszer, a www3 kétszer, a www4 ismét egyszer, végül a www5 kapja a maradék kettőt. A *servers.map* lista finomhangolását valós időben végezhetjük az alkalmazáskiszolgálókon futtatott `top` programok figyelése közben (avagy a `tl` alkalmazásával, ahogy azt a [#59] „Az átlagos terhelés folyamatos megjelenítése a címsorban" trükkben megismerhettük). Az Apache minden egyes találatnál ellenőrzi a fájlhoz tartozó `mtime` értéket, és amennyiben időközben megváltozott volna, újraindítás nélkül, önműködően újratölti. Ezért aztán egy intelligens rendszerfelügyeleti eszköz figyelheti az összes *www**kiszolgálót, és ha valamelyik nehézségekkel küzd, kiszedheti a listából. Megfelelően óvatos tervezéssel olyan webszolgáltatást is létrehozhatunk, amely soha nem áll le, mivel a gondot okozó kiszolgálókat kidobja, s mindezekhez soha nem kell újraindítanunk az Apache-ot sem.

Több alkalmazáskiszolgálóval egyszerre dolgozni nem könnyű feladat. Természetesen a fájlrendszereinket össze kell hangolnunk, NFS és esetleg az `rsync` segítségével, ahogy azt a [#59] „Az átlagos terhelés folyamatos megjelenítése a címsorban" trükkben leírtuk. De még a legkörültekintőbben megtervezett rendszereknek is vannak kényes pontjai. Mi lehet a gond, ha *Internal Server Error* hibát kapunk, de egy frissítés után eltűnik? Valamelyik alkalmazáskiszolgálónk meghibásodott, és sajnos sehogyan sem tudjuk megmondani, melyik volt az (hacsak végig nem nézzük az összes gép hibanaplóját). Nézzünk egy hasznos kis újraíró szabályt, amellyel pontosan megadhatjuk, melyik alkalmazáskiszolgálóra szánjuk az adott üzenetet:

```
:
RewriteCond %{ QUERY_STRING}  appserver=(.*)
RewriteRule ^/(.*) http://%1.oreillynet.com/$1
➥ [P,L]
```

Ettől kezdve bármelyik alkalmazáskiszolgálót elérhetjük, ha a böngészőnk címsorába a következő sort gépeljük be:

```
http://www.oreillynet.com/some/path/?appserver=www3
```

Ezzel a módszerrel hamar felderíthetjük az esetleges hibákat. Előtét proxykiszolgálókkal, címforgatásos DNS-szolgáltatással és adatbázismásolatokkal együtt egy egyszerű, de hatalmas telepítésekhez is megfelelő beállításhoz juthatunk.

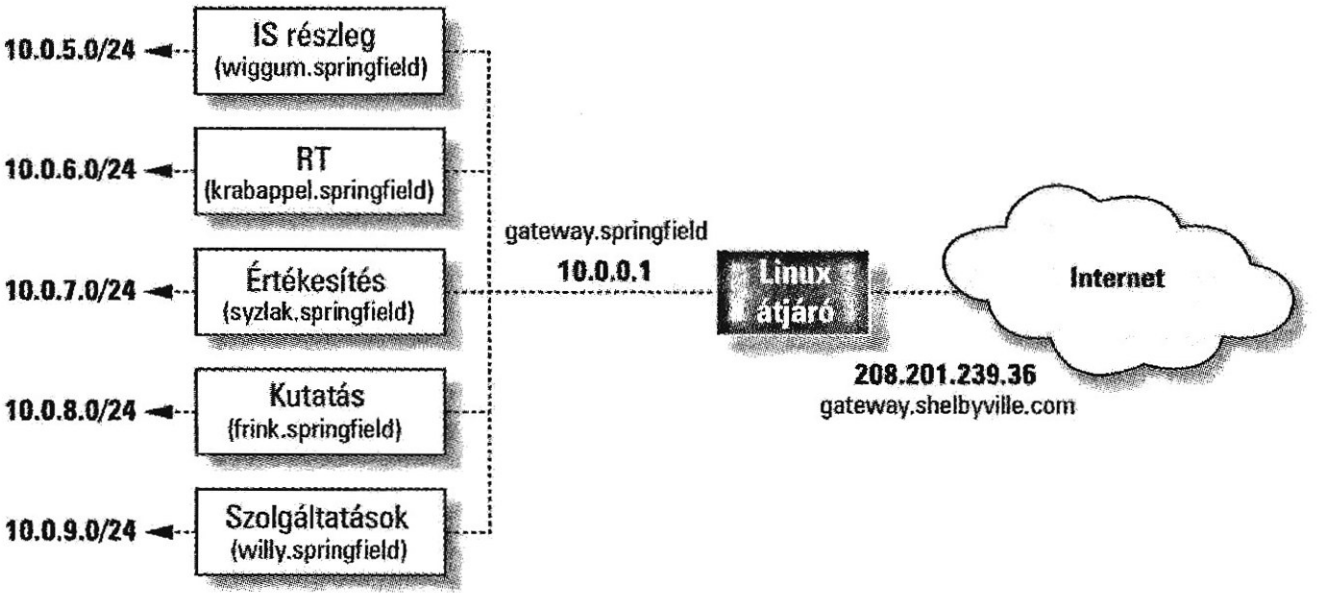
## Kapcsolódó anyagok

- Az Apache `mod_rewrite` útmutatóját a `http://httpd.apache.org/docs/mod/mod_rewrite.html` címen találjuk.
- Proxy Server Hack
- [#38] rsync az SSH-n keresztül
- [#79] Terhelésmegosztás címforgatásos DNS segítségével
- [#82] Másolatkészítés MySQL-lel

# 100 Szuperszolgáltatás: tömeges webhelyszolgáltatás helyettesítő karakterekkel, proxyval és újraírással

## Tartsuk fenn belső webkiszolgálók ezreit anélkül, hogy akár a kisujjunkat mozdítanánk

Tegyük fel, hogy egy nagy méretű belső hálózatot kezelünk, amely NAT-útválasztó mögé rejtettünk az internet többi része elől. Hálózatunk felépítése a 8.2. ábrán látható.




### 8.2. ábra

*A vállalati hálózatok általában belső címzést alkalmaznak, és legalább egy internet átjáróval rendelkeznek, ami a hálózati címváltást (Network Address Translation) végzi*

Tegyük fel, azt szeretnénk, ha a saját hálózatunkon bárki felállíthatná saját webkiszolgálóját. De, mint minden jó rendszergazda, elég ravaszak és lusták vagyunk ahhoz, hogy ne akarjunk folyton újraírásszabályokkal vacakolni, valahányszor valaki változtatni akar valamit. Meggondoltan és óvatosan használva a virtuális gazdákat, valamint a `mod_proxy` és `mod_rewrite` modulokat, teljes hálózatunk felügyeleti igényét egyszerű DNS-bejegyzésmódosításokra csökkenthetjük. Ezután már nem sok tarthat vissza bennünket attól, hogy a felelősséget gyorsan áthárítsuk arra a szervezeti egységre, amely mindenáron webkiszolgálókat akart.

Először is az átjárógépen szükségünk lesz egy működő `mod_rewrite` és `mod_proxy` támogatással bővített Apache-ra. Szükségünk lesz továbbá saját felsőszintű belső tartományszolgáltatást nyújtó DNS-re (amit a [#80] „Saját felsőszintű tartomány üzemeltetése" trükkben ismertettünk). Tételezzük fel, hogy a saját tulajdonú *shelbyville.com* tartományt használjuk, valamint a belső gépeinkhez már beállítottuk a *.springfield* belső felső szintű tartományt.

Adjuk a következő sorokat az átjáró Apache beállításaihoz:

```
Port 80

BindAddress *
NameVirtualHost *

<VirtualHost *>
ServerName mux.shelbyville.com
ServerAlias *.shelbyville.com

RewriteEngine On
RewriteCond %{ HTTP_HOST}   (.*).shelbyville.com
RewriteRule ^/(.*) http://%1.springfield/$1 [P]
</VirtualHost>
```

A fenti beállítás röviden a következőket jelenti:

- figyelj minden elérhető IP-címre a 80-as kapun,
- fogadd el a *.shelbyville.com* tartományba tartozó gépeket célzó kérelmeket,
- a kért HTTP-gazda nevéről válaszd le a *.shelbyville.com* részt,
- hozz létre kapcsolatot (proxyn keresztül) a *.springfield* utótaggal kiegészített megmaradó névvel,
- küldd el az eredeti URI-t, majd az eredményt add vissza az ügyfélnek.

Mindezt azért tehetjük meg, mert HTTP 1.1 alatt a gazdagépeket név szerint adjuk meg a HTTP-fejlécben, és így nincsenek állandó IP-címhez kötve. Az Apache a rendszer névfeloldóval megkeresteti a `%1` tartalmát (minden, ami a *.shelbyville.com* előtt található), majd meg-

próbál kapcsolódni hozzá. Mivel az átjáró olyan DNS-kiszolgálót használ, amely ismeri a *.springfield* felső szintű tartományunkat, a megfelelő belső gazdát fogja proxyként használni.

Tegyük fel, az eredeti kérelem a *http://jimbo.shelbyville.com/index.html* volt. A `RewriteCond` sor után a `%1` csak a jimbo szót tartalmazza. Ezután megpróbálunk proxykapcsolatot létrehozni a *.springfield* utótaggal kiegészített `%1` (jimbo) névvel. Mi lesz az eredménye? Proxykérelem indul a *jimbo.springfield* nevű gépre, majd átadódik az eredeti URI-tartalom, mintha az átjáró soha nem is létezett volna.

Ez a legegyszerűbb beállítás, de sajnos így a sütiket használó belső kiszolgálókkal még nem boldogulunk. Amennyiben a belső hálózaton is szeretnénk sütiket használni, próbáljuk ki a következő beállítást:

```
<VirtualHost *>
ServerName mux.shelbyville.com
ServerAlias *.shelbyville.com

RewriteEngine On

RewriteCond %{ HTTP_HOST}   (.*).shelbyville.com
RewriteRule (.*) $1 [E=WHERETO:%1.springfield]

ProxyPassReverse / http://%{ ENV:WHERETO} /

RewriteRule ^/(.*) http://%{ ENV:WHERETO} /$1 [P]
</VirtualHost>
```

Itt egy „hamis" `RewriteRule` utasítást használunk, amit csakis azért hívtunk meg, hogy értékkel töltse fel a `%{WHERETO}` környezeti változót. Ebbe kerül bele az eredetileg igényelt HTTP-gazda neve, miután megfosztottuk a *.shelbyville.com* végződéstől és bővítettük a *.springfield* utótaggal. Erre azért lesz szükségünk, hogy a módosított gépnevet átadhassuk a `ProxyPassReverse`-nek.

A *.shelbyville.com* és *.springfield* tartományok DNS-beállításainak megváltoztatásával könnyedén indíthatjuk és leállíthatjuk a webkiszolgálókat, anélkül, hogy az átjáró Apache beállításain bármit is vál-

toztatni kellene. Amennyiben még kényelmesebbé szeretnénk tenni a dolgokat, a *.shelbyville.com* tartományhoz használhatunk helyettesítő karakteres DNS-t is:

```
*.shelbyville.com IN A 12.34.56.78
```

(Természetesen, az 12.34.56.78 helyére az átjárónk külső IP-címe kerül.) Mostantól az átjáróba kerülő minden *barmi.shelbyville.com* alakú kérelem a nélkül továbbítódik, hogy a zónafájlt módosítanunk kellene. Most már kizárólag a belső DNS-beállításokat (*.springfield* tartomány) kell nekünk kezelnünk. A legegyszerűbb módszer, amivel ezt a felelősséget is elháríthatjuk magunkról, ha több altartományra bontjuk, amelyek más belső DNS-kiszolgálókra mutatnak. Például valami ilyesmit helyezünk a *named.conf* állományba:

```
zone "wiggum.springfield" {
type slave;
file "wiggum.db";
masters {  10.42.5.6; } ;
} ;

zone "krabappel.springfield" {
type slave;
file "krabappel.db";
masters {  10.42.6.43; } ;
} ;

zone "syzlak.springfield" {
type slave;
file "syzlak.db";
masters {  10.42.7.2; } ;
} ;
```

A listát tetszőleges hosszan folytathatjuk. Mostantól minden egyes részleg saját DNS-kiszolgálóval rendelkezik, amelyben új gépeket (és így egyben internetkész webkiszolgálókat) vehet fel, s ehhez még csak egy elektronikus levelet sem kell váltanunk. Az új kiszolgálóinkat az interneten keresztül a *http://rudi.wiggum.shelbyville.com/* címen érhetik el, amely belül a *http://rudi.wiggum.springfield/* címre

fordítódik, amit végül a *wiggum* tartományért felelős részleg DNS-kiszolgálója keres ki. Az internetfelhasználók szempontjából valamennyi válasz közvetlenül az átjáróról érkezik, és még csak nem is tudják, hogy a *.springfield* tartomány egyáltalán létezik.

Most már csak az a kérdés, mi a csudát fogunk kezdeni a rengeteg megtakarított időnkkel?

# Tárgymutató



## A, Á

## B



## C, Cs



## D



## E, É

## F



## G, Gy



## H



## I



## J



## K

## L



## M



## N, Ny



## O, Ö



## P

## Q



## R



## S, Sz

## T, Ty



## U, Ü



## V



## W

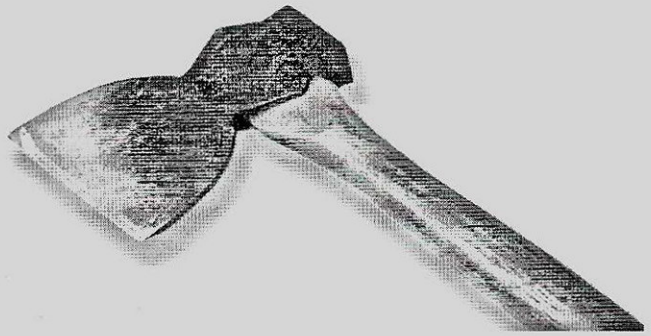


## X



## Z

# LINUX BEVETÉS KÖZBEN



A számítógépes rendszereket, az internetet és a vállalati kiszolgálókat karbantartó, a felmerülő gondokat és kihívásokat ötletesen és hatékonyan megoldó, örök újító és felfedező informatikusokat hívja a szakma szakiknak, betyároknak, angolul „hacker"-eknek. A szakma által elismert szakembereknek tehát mind tudással, mind tapasztalattal, mind pedig a világ jobbá tételére irányuló vággyal rendelkezniük kell. E szakemberek szerteágazó feladatokat oldanak meg, alkotnak, hisznek a szabadságban és a kölcsönös segítségnyújtásban. Rob Flickenger célja, hogy segítsen mindenkinek, aki követni kívánja a betyárok által megálmodott utat.

Igazán hatékony munkát csak az végezhet, aki minden olyan nehézséget képes megoldani, amivel a rendszer – a bekapcsolástól a leállításig – meglepi. A szerző a közbeeső idő átvészeléséhez nyújt segítséget a mindennapos rendszerfelügyeleti feladatok újszerű és időt megtakarító megközelítésével, mindez azonban nem bevezetés a rendszerfelügyeletbe, hanem olyan hatékony és ötletes módszerek gyűjteménye, amelyeket még a tapasztaltabb rendszergazdák sem mindig vesznek észre.

A könyvben szó esik a rendszerindítási folyamatba való beavatkozástól kezdve a parancssorral végzett hatékony munkavégzésen, a gyakori feladatok önműködővé tételén, a változatkövetésen és a hálózatkezelésen át a biztonsági kérdések tárgyalásáig számos, mindennapi feladatot segítő tippről. Ezenkívül a kiszolgálók három nagy alkalmazását is megismerhetjük: a BIND 9-et, a MySQL-t és az Apache-t. Olyan trükkök birtokába juthatunk, amelyekkel gyorsabban és hatékonyabban lehet e szolgáltatásokat üzemeltetni – anélkül tehát, hogy sok munka lenne velük.

KISKAPU
O'REILLY®

**Kategória:** operációs rendszerek, Linux
**Felhasználói szint:** haladó
ISBN szám: 963 9301 55 8
**Ára: 3500 Ft**

ISBN: 963 9301 55 8
9 789639 301559