E-közigazgatási Szabad
Szoftver Kompetencia Központ

Mátó Péter, Rózsár Gábor, Őry Máté,
Varga Csaba Sándor, Zahemszky Gábor

20/80

Unix és Linux alapismeretek rendszergazdáknak

ÚJ SZÉCHENYI TERV

20/80 – Unix és Linux alapismeretek rendszergazdáknak

Módszertan és tartalmi tervek: Mátó Péter, Varga Csaba Sándor, Zahemszky Gábor

Írták: Mátó Péter, Rózsár Gábor, Őry Máté, Varga Csaba Sándor, Zahemszky Gábor:

0.9-es változat, 2014. április 11., elektronikus kiadás

Kiadó: E-közigazgatási Szabad Szoftver Kompetencia Központ

Honlap, javított kiadások: http://szabadszoftver.kormany.hu/sajat-oktatasi-anyagok/.

ISBN 978-963-08-8298-9

Szerzői jog

Tartalomjegyzék

Előszócska

Ezzel a könyvvel és testvéreivel az a célunk, hogy viszonylag tömören összefoglaljuk azokat az információkat, amiket egy szabad szoftvereket használó szakembereknek tudnia illik.

20/80. Mit akar ez jelenteni? Tapasztalatunk szerint a létező eszközöknek és információknak csak egy kis része szükséges a mindennapok tipikus feladatainál. Igyekeztünk kiválogatni nektek a tudásnak azt a 20%-át, ami az általában előforduló feladatok 80%-ánál elegendő lesz. Célunk ezen elv alapján összeszedni, rendszerezni és átadni a leghasznosabb dolgokat. Hiába próbálnánk mindent elmondani – nekünk nincs időnk mindent leírni, nektek meg nincs időtök elolvasni. Ezért sok minden kimarad. Ha úgy gondolod, hogy fontos, kimaradt vagy bővebben kellene beszélni róla, szólj! Ha valami hibás, szólj! E-mail címünk: esz2k2@gmail.com. De ha írsz, légy türelemmel, valószínűleg 200 másik levél is vár még megválaszolásra. A továbbfejlesztés során minden konstruktív javaslatot igyekszünk majd az anyagba építeni.

A tárgyalt megoldások és szabad szoftverek legtöbbször több operációs rendszer alatt is használhatóak. Amikor viszont operációs rendszer szintről esik szó (telepítés, csomagkezelés vagy finomhangolás), akkor ez most – népszerűsége miatt – nálunk Linuxot jelent.

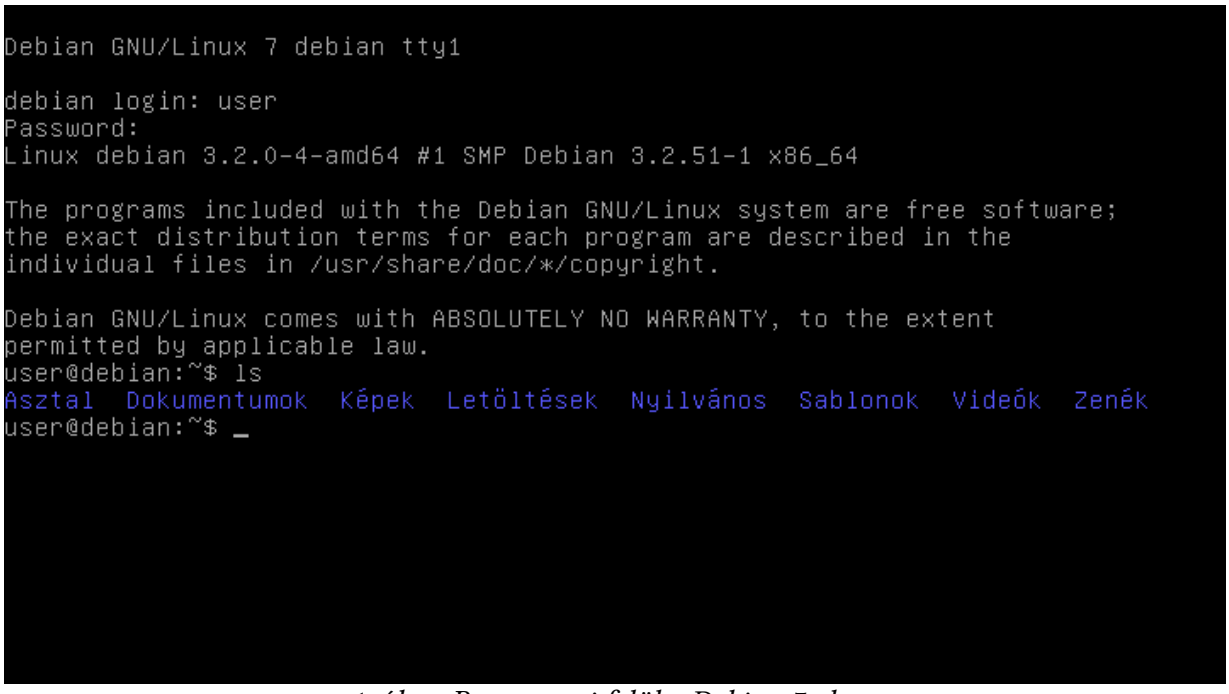
A könyvben időnként kérdéseket teszünk fel, de néha nyitva hagyjuk a választ. A cél: gondolkozz, olvass utána, használd az agyadat! Ha egy témát alaposabban meg akarsz ismerni, akkor nincs mese, alaposabban utána kell olvasnod. Minden területnek megvannak a maga – tőlünk sokkal mélyebb ismereteket tárgyaló – szakkönyvei, előttük azért érdemes a mi összefoglalónkat elolvasni, mert ezekben – reményeink szerint – az adott terület esszenciája található. Ez alapján már könnyebben eligazodsz majd a 6-700 oldalas, lényegesen kimerítőbb anyagokban is.

Alapfogalmak: parancssor, grafikus felület, munkakörnyezetek

A Linux rendszerekkel a legkülönbözőbb helyeken találkozhatunk: szuperszámítógépeken, nagygépeken, szervereken, hálózati- és táreszközökön, asztali gépeken, notebookokon, ipari eszközökön, mobiltelefonokon, vagy éppen a hűtőszekrényen. Számos tekintetben azonos elemekből építkeznek ezek a rendszerek, azonban a végfelhasználó mindezeket különböző felületeken éri el. Ezen felületek közül kiemelkedik a parancssori felület, ami valamilyen formában minden gépen elérhető, általában képernyőn és billentyűzettel, hálózaton, vagy soros vonalon keresztül.

A fogyasztói eszközök (PC-k, táblagépek, mobiltelefonok) általában grafikus felületen érhetőek el. Ezen belül a PC-ken használt felületek alapeszköze az ablakozó rendszer, amely lehetővé teszi azt, hogy egyidejűleg több alkalmazást futtassunk, és azok között váltsunk. A táblagépek és a mobiltelefonok ezt a problémát máshogy oldják meg. Végül nem szabad megfeledkezni azokról az eszközökről sem, amelyekre közvetlenül képernyőt nem csatlakoztatunk, hanem más perifériáikkal, egyedi módon kommunikálnak a felhasználóval.

A parancssor



1. ábra. Parancssori felület Debian 7 alatt.

A parancssor interfész (command-line interface, CLI) a Unix eredeti felhasználói felülete, a mai napig szinte minden rendszernek része. Ezen a felületen parancsokat gépelhetünk be, amelyeket a parancsértelmező (shell) futtat. Az alapbeállítás szerinti parancsértelmező a bash, de sokan kedvelik a csh-t, a ksh-t és a zsh-t is.

A parancssori felületet számos módon érhetjük el. A grafikus felület nélkül telepített Linux rendszereken általában a gép indítása után a login alkalmazás fogad minket, amely felhasználónevünket és jelszavunk megadása után elindítja a parancsértelmezőt (1. ábra). (Ne ijedjünk meg, ha a begépelt jelszavunk nem látszik, ez biztonsági okokból van így.)

Ez a felület egy virtuális terminál. Az elnevezés a nagygépes Unix rendszerek korából származik, amikor a felhasználók soros vonalon, terminálokon keresztül csatlakoztak a géphez. Ez a felület ennek „virtuális” megvalósítása a gépre kötött megjelenítővel és billentyűzettel. Nem csak egy ilyen virtuális terminált érhetünk el, általában hat ilyen vonalat érhetünk el (lásd még a Rendszerindítás című fejezetet), amelyek között az Alt+sorszám billentyűkombinációkkal válthatunk. Akkor is elérhetőek a virtuális terminálok, ha a rendszerünkön grafikus felület is fut: a Ctrl+Alt+sorszám kombinációval.

A parancssori felületet rugalmassága lehetővé teszi azt is, hogy számos más módon csatlakozzunk hozzá. Gyakran érjük el hálózaton, az SSH protokollon keresztül (bővebben a Távoli elérés: ssh című fejezetben). Lehetőségünk van a grafikus felületről is a gép parancssori felületét elérni, amelyhez terminálemulátorra lesz szükségünk.

Grafikus felület

A grafikus felület napjainkig elterjedt megvalósítása az X11 keretrendszer. Ez egy kliens-szerver felépítésű architektúra, amelyben a szerver vezérli a megjelenítőt, míg a kliensek az egyes alkalmazások, amelyek ablakokat kívánnak megjelentetni. A legtöbb esetben a kliens és a szerver egyazon helyen fut: a képernyő arra a gépre van kötve, amely az alkalmazásokat futtatja. Ez a felépítés azonban lehetővé teszi azt, hogy különböző okokból távol futó alkalmazásokat használjunk grafikusan.

Az X11 referenciaimplementációja az X.Org Server, amely egyben a legnépszerűbb is. A nyolcvanas évek óta számos örökséget magával hordó X11 szabványt több alkalommal igyekeztek kiváltani. A jelenleg is aktív törekvések a Wayland és a Mir.

Ablakkezelők, asztali környezetek

Az X11 keretrendszer szolgáltatásai korlátozottak, kényelmes grafikus felhasználói felület eléréséhez ablakkezelőre is szükségünk van. Számos választási lehetőségünk van, azonban a legnépszerűbb ablakkezelőket a még több szolgáltatást nyújtó asztali környezetek részeként érhetjük el.

Az asztali környezetek közti választás ízlés kérdése, azonban a legtöbb Linux-disztribúció döntött egy-egy környezet mellett, amelyet kiemelten támogat.

A közelmúltig két nagy asztali környezet osztotta meg a Linux-felhasználókat: a GNOME és a KDE. Ezen kívül jelentős, egyszerűbb alternatíva az XFCE. Ezen a piacon jelent meg az Ubuntu új felülete, a Unity, amely nagyrészt a GNOME alkalmazásaira támaszkodik.

GNOME

A GNOME az egyik legnépszerűbb asztali környezet. Számos disztribúció támogatja kiemelten különböző változatait.

A GNOME 2 hosszú ideig egységes képet adott a különböző disztribúcióknak. A Red Hat Enter-

prise Linux és a CentOS aktuális verzióiban, valamint a Mate projektben találkozhatunk vele.

A GNOME 3 gyökeresen megújította a felhasználói felületet, amelyet már kevesebb disztribúció fogadott el. Az alaptelepítés része például Fedora alatt.

Egyéb rendszerek

A nagy asztali környezeteken kívül számos ablakkezelőt érhetünk még el, amelyeket a legtöbbször haladó felhasználók használnak. Kiemelhető két fő irány: a hagyományos lebegő ablakokat használó minimalista ablakkezelők (stacking window manager, például Openbox, Blackbox), valamint a csempéző ablakkezelők (tiling window manager, például awesome, i3, xmonad).

Grafikus belépés

Ha számítógépünket indításától kezdve grafikusan szeretnénk használni, akkor egy bejelentkezéskezelőre (login manager) is szükségünk van. Ebből is számos alternatíva érhető el, azonban a grafikus felülettel érkező disztribúciók szintén leveszik vállunkról a választás terhét.

A bejelentkezéskezelők közül kiemelhető a GNOME-hoz kötődő GDM; a KDE részét képező KDM; valamint a GDM kiváltására készített Lightdm.

Miért beszélünk egyáltalán parancssorról?

A Linuxot ellenzők sok egyéb mellett rendszeresen hátrányként hozzák fel azt, hogy nincs egységes kezelőfelület, amin keresztül mindent – de legalábbis az adminisztrációs munka nagy részét – könnyedén, egy-két kattintás segítségével el lehetne végezni. Lehet a dolgot szépíteni, de fölösleges. Ma linuxos szerver üzemeltetése során nagy eséllyel lesz egy olyan pont, amikor a webböngészőn keresztüli adminisztratív eszköz nem elegendő, vagy az adott terjesztéshez elkészített (nem feltétlenül) webes adminisztrációs program egy adott feladatot már nem képes elvégezni – pl. mert annyira nyakatekert dolgot szeretnénk megcsinálni, amire a szoftver fejlesztői nem gondoltak. Ezt lehet felróni, de ha Linux kiszolgáló (vagy akár Linux munkaállomás) használatára vetemedik valaki, jobb ezt az elején tisztázni. Nem nagyon lehet megúszni a parancssort.

Bejelentkezés

Egy linuxos gép használatához a gépre be kell jelentkezni. Egy gépet elérhetünk közvetlenül a gép előtt állva (ülve) és hálózaton keresztül távolról is, illetve egy másik csoportosítás szerint használhatjuk buta, karakteres felületen, vagy a mai rendszerekre jobban hasonlító grafikus felületen keresztül is (amely amúgy szintén lehet akár lokális, akár távoli elérés). Főleg karakteres felületű elérés esetén a bejelentkezéshez általában *felhasználói név* és *jelszó* szükséges – ezeket eredendően a rendszergazda állítja be, akinek hagyományosan root a bejelentkezési neve. (Grafikus felületen sok esetben nem felhasználói nevünket kell megadni, hanem valódi nevünket kell egy listából kiválasztani – de a háttérben a rendszer ilyenkor is felhasználói nevünkkel azonosít minket.) Sikeres bejelentkezés után már dolgozhatunk is.

Alapparancsok

A továbbiakban hosszabb-rövidebb részekre lebontva elmagyarázzuk a parancssor használatának alapvető építőkockáit, amit aztán mindenki igényének megfelelően használhat. Szó lesz ezen kívül néhány egyszerű, a napi használat során sűrűn előforduló, vagy ritkábban használt, de fontos egyéb eszközről is. Hozzáolvasni nem muszáj, de sok esetben kifejezetten javasolt.

A parancsértelmező

A Linux szellemi elődjének számító Unix rendszerekben az idők során eléggé sokféle parancsértelmező jött létre, de a legtöbb Linux-terjesztés a bash névre hallgató (Bourne Again SHell)[1] nevűt adja alapból. Azaz, ha valamilyen módon karakteres felületű hozzáféréshez jutunk Linuxunkon[2], szinte biztos, hogy ez a parancsértelmező indul el. A napi beszédben egyszerűen csak *shell*ként

1. A név utalás a korai shellek közül legsikeresebbnek minősülő – kvázi szabványként elterjedt, Steve Bourne által fejlesztett sh-ra (amit pedig Bourne-shellként szoktak emlegetni).

szoktunk hivatkozni a parancsértelmezőre, bár létezik magyar nyelvű szakirodalom, amely a jóval kevésbé elterjedt – de kétségtelenül legalább magyarnak tűnő – héj, vagy burok kifejezéssel hivatkozik erre az eszközre[3].

Parancssor

Elsődleges feladata a különböző parancsértelmezőknek az általunk begépelt parancsnak és a parancstól és egymástól jellemzően szóközökkel, tabulátorokkal elválasztott opcióknak (módosítók) és egyéb paramétereknek a feldolgozása, értelmezése; egyszerűbben az általunk begépelt utasításoknak a végrehajtása. A parancsokat egyszerűbb esetben az ENTER (soremelés) karakter zárja le. A parancssor általános felépítése:

```
$ parancs [ -o -m --hosszú-opció] [ fájl | könyvtár | gép | felhasználó |
  egyéb ]
# parancs [-om --hosszú-opció] [ fájl | könyvtár | gép | felhasználó | egyéb ]
```

A példákban a parancssor elején egy ún. prompt – a parancsra várás jele – áll, ezt begépelni sosem kell. Hagyományosan a $-jel azt jelenti, hogy tetszőleges felhasználó végrehajthatja az adott parancsot, a #-jel pedig azt, hogy rendszergazda jogosultság kell a futtatáshoz. A „-o” és „-m” opciók (rövid forma) és a „--hosszú-opció” a műveletet magát valamilyen formában befolyásoló ún. módosító paraméterek. A rövid, egybetűs opciók többségében írhatók egyben is (mint a 2. példában), de hosszú formájukban megadva külön-külön kell írni őket. Az esetek többségében a módosítók megadási sorrendje nem számít – de erről minden esetben a konkrét parancs dokumentációjában érdemes meggyőződni; viszont ajánlott a fent jelzett formában az egyéb paramétereket (pl. gépek, felhasználók, fájlok neve) a parancssorban az összes módosító után, a parancs végén megadni, és nem bekeverni az opciók közé.

Írhatunk egy sorba több, egymás után végrehajtandó utasítást is, ebben az esetben az egyes parancsokat a ; (azaz a pontosvessző) választja el egymástól:

```
$ parancs1 parm1 parm2 ; parancs2 parm3 parm4;parancs3
```

Mint látható, nem kell, de lehet a ; jobb- és baloldalán szóközöket írni, ezt általában az olvashatóság növelésére szokták használni. Fenti parancssorban először a parancs1, amikor befejeződött a futása, akkor parancs2, és amikor az is befejeződött, akkor parancs3 indul el. (Pont ugyanúgy, mintha a fenti három parancsot külön sorba írtuk volna.)

Néhány egyszerű parancs

Néhány egyszerűbb, a mindennapokban jól használható parancs:

```
exit
```

2. Ez lehet a gép karakteres konzolja, egy távoli SSH-hozzáférés, egy titkosított csatornán keresztüli telnet-elérés (titosítás nélkül a telnet erősen ellenjavalt), vagy akár a grafikus felületről elérhető, ún. terminál által nyújtott hozzáférés.
3. Noha elvben egyetértünk azzal, hogy amit lehet, magyarul kell mondani, bizonyos esetekben kifejezetten ellenségesen viszonyulunk néhány – jelenleg nem túl elterjedt – „magyarításhoz”. És noha az adott könyvek írójának, fordítójának a szakmai munkáját nem akarjuk leszólni (nem is lenne miért), maradunk ennél a megközelítésnél – ha számunkra is elfogadhatónak tűnik a magyar kifejezés, akkor azt, ellenkező esetben kíméletlenül az angol nyelvű eredeti kifejezést használjuk.

A parancsértelmező futását befejezi, bizonyos esetekben a teljes rendszerből való kijelentkezést is jelent[4]. Interaktív shell esetén sokszor helyette egy üres sor elején a Ctrl-D billentyűkombináció lenyomását használjuk ugyanerre a funkcióra.[5] Ha már szóba került a kijelentkezésre is használható billentyűkombináció, még párat hadd említsünk meg: a Ctrl-C lenyomásával megszakíthatjuk az éppen futó alkalmazást; ritkán szükség lehet ennek a durvább formájára, a Ctrl-\ (igen, a Control mellé a hanyatt-törtvonalat kell nyomni); illetve meglehetősen kényelmessé teszi az életet, ha rájövünk a négy „nyílgomb" funkciójára.

```
man parancsnév
```

A paraméterként megadott „parancsnév" használatát bemutató ún. kézikönyvlap megjelenítésére szolgál. Az egyik legfontosabb parancs, sajnos szeretik elfelejteni, hogy a linuxos parancsok általában elég jól dokumentáltak. Van egy rendszeresen elfelejtett funkciója: a kézikönyvlapokon szereplő információkból a rendszer általában létrehoz egy speciális adatbázist, amely adatbázisban kulcsszó alapon kereshetünk, ezzel sok esetben akkor is rátalálhatunk egy parancsra és dokumentációjára, ha magát a parancs pontos nevét nem is ismerjük. Erre szolgál a -k opció:

```
man -k kulcsszó
```

A dokumentációt hagyományosan fejezetekre bontják, az egyes fejezeteket pedig jellemzően számokkal „nevesítik". Normál (értsd: extra jogosultsággal nem rendelkező) felhasználók számára elérhető parancsok dokumentációja az 1-es számú, a csak rendszergazdák számára elérhető parancsok pedig hagyományosan a 8-as számú fejezetben találhatók, elsősorban tehát az itt található információk érdekesek (persze a közbenső fejezetszámok is léteznek, és az ott található információk is hasznosak lehetnek). Ezt a fejezetszámozást azért is érdemes ismerni, mert szokás ezekre a dokumentációkra hivatkozni, a hivatkozás formája: man(1) – ami azt jelzi, hogy az egyes számú fejezetben szereplő dokumentációról beszélünk, méghozzá a man parancshoz tartozóról. A man parancs meghatározott sorrendben keres a fejezetek között. Esetenként szükség lehet egyértelműsíteni, hogy nem minden, hanem csak egy meghatározott fejezetben szeretnénk keresni, ekkor a parancsnak meg kell adnunk magát a fejezetazonosító számot is:

```
man 4 passwd
```

Mindegyik fejezet jól meghatározott részét írja le a rendszernek, hogy pontosan mit, az kideríthető az egyes fejezetekhez tartozó ún. intro kézikönyvlap elolvasásával. (Pl. man 4 intro és man 1 intro elolvasása után kitalálható, hogy a man passwd – ami amúgy a man 1 passwd paranccsal ekvivalens -, és man 4 passwd között mi a különbség.) Levelezőlistákon, fórumokon feltett kérdések esetén meglehetősen gyakori az RTFM válasz – a rövidítés kifejtése: Read That F.. Manual[6] – azaz Olvasd el azt a … kézikönyvet! Ez a velős „nemválasz" általában azt jelzi, hogy a kérdező a legalapvetőbbnek tekinthető lépést sem tette meg a probléma megoldása érdekében: még a dokumentációt sem olvasta el – ha megtette volna, megtalálta volna a választ[7]. Ez a rövid kifakadás is jelzi a fentebb már szereplőt: tessék elolvasni a dokumentációt, azért van. (Ha már rövidítés, elsősorban triviális hibák elkövetésekor kapható még a PEBKAC[8] válasz is – nem sokkal szebben utal az elkövető hibáira.)

4. Ha karakteres felületen jelentkezünk be (a korábban emlegetett konzolos, vagy távoli: SSH/telnet), akkor az exit jellemzően kijelentkeztet a rendszerből, ha grafikus felületről indítottunk egy terminálemulátort, akkor általában csak az adott terminálablak bezárása történik meg.
5. Részletesebb információért olvasd el az stty parancs dokumentációját, és legalább az EOF beállítását (illetve a többit is, ha a Ctlr-C, … is érdekel)
6. Maradjunk abban, hogy az F a Funny rövidítése (nem, nem az)
7. Gyakorlatilag az RTFM a lmgtfy.com internet-előtti, pre-Google megfelelője
8. Problem Exists Between Keyboard and Chair – azaz a hiba forrása a billentyűzet és a szék között keresendő; gy. k.: maga a műveletet végrehajtó (aki valószínűleg a kérdést feltevő) rontott el valamit

Nagyon sok linuxos parancs fejlesztője a man helyett egy másik eszközt, az ún. info fájlokat használja dokumentálásra. Ezért érdemes az info (esetleg pinfo) nevű parancsnak is utánanézni.

Áttételesen ugyan, de már szerepelt a példában, így nézzük a következő fontos parancsot:

`passwd`

Ez szolgál a bejelentkezési jelszó átállítására. Rendszergazdák más felhasználók jelszavát is módosíthatják a passwd felhasználóinév forma használatával.

`echo`

Segítségével üzenetet jeleníthetünk meg; nyilván mindennapos munkánk során nem várjuk, hogy a gép hülyeségeket fecsegjen (pláne utasításra), de pl. parancsfájlokban meglehetősen gyakori a használata.

`clear`

A parancs letörli a képernyőt.

`date`

A nyelvi beállítások függvényében az aktuális dátum, idő, időzóna információk megjelenítésére szolgál.

`who`

Megmutatja, hogy az adott pillanatban kik vannak a gépre bejelentkezve, ők hol és mikor jelentkeztek be.

Záróakkord

Ez a jéghegy csúcsa, illetve az ott fagyoskodó hangya bal hátsó lába. Aki hatékonyan szeretne Linux, Unix rendszerekkel foglalkozni, az lehetőleg minden parancsnak nézzen utána, de borzasztó címe ellenére olvassa el „A shell programozása minimalista megközelítésben” című részt is.

Felhasználók és csoportok, Unix típusú rendszer használata

Minden Linux rendszerben vannak felhasználók. Lehet, hogy csak egy – a rendszergazda -, ennek ellenére a fogalommal és az ehhez kapcsolódó információkkal tisztában kell lenni.

Egy felhasználóra a rendszer két különböző módon hivatkozhat: vagy bejelentkezési név (loginname, logname) alapján – ezt jellemzően valamilyen karakteres felületű bejelentkezéshez, különböző információk megjelenítéséhez használja a rendszer; vagy pedig egy nem-negatív egész szám, egy egyedi felhasználói azonosító (angolul: user identifier, rövidítve UID) használatával. Ez utóbbit használja a rendszer belül, az adatok tárolásakor, ellenőrzéskor. Hagyományosan a root bejelentkezési névhez tartozik a 0-s UID – ő a rendszer adminisztrátora; jellemzően mindenhez jogot biztosít, ha valaki root-ként tevékenykedik. (Általában az a jellemző, hogy a logname / UID hozzárendelés kölcsönösen egyértelmű, de ritkán előfordul, hogy több bejelentkezési név mögött a valóságban ugyanaz az UID áll. Ez utóbbi nem javasolt beállítás.)

A felhasználókat csoportokba sorolják. A felhasználói csoportoknak szintén van neve, a csoportnév (groupname) teljes mértékig független a felhasználói névtől, bár jellemző, hogy vannak azonos nevű felhasználók és csoportok (és ha ez fennáll, nagy eséllyel a felhasználó tagja az azonos nevű csoportnak). A csoportokat szintén számmal azonosítja a rendszer, ez a csoportazonosító (angolul group identifier, röviden GID). A 0-s UID-hoz (root felhasználó) hasonlóan létezik 0-s GID is, szintén root (csoport)névvel[9]. Hagyományosan a 0-s csoporthoz tartozás extra jogosultságot biztosított, ma ez többségében már csak csak extra konfiguráció után érhető el.

A felhasználói és csoportnevek kiválasztása, az UID-ek és GID-ek kiosztása, a felhasználók csoportba sorolása a rendszergazda feladata. Ezek az adatok alapértelmezetten egy /etc/passwd és egy /etc/group nevű kötött formájú szöveges fájlban tárolódnak. A fájlok szerkeszthetők akár kézzel is, az erre szolgáló vipw és vigr parancsokkal (csak óvatosan!), de léteznek parancssoros eszközök: a felhasználók kezelésére a useradd/usermod/userdel parancshármas, míg a csoportok kezelésére a hasonló nevű groupadd/groupmod/groupdel szolgál.

Linux használatához be kell jelentkezni a rendszerbe. Karakteres felületen ezt tipikusan a bejelentkezési név és a hozzá tartozó jelszó megadásával lehet megtenni, grafikus felületen ellenben sokszor a felhasználókhoz tartozó valódi nevet tartalmazó listából választjuk ki a megfelelőt, esetleg nem is jelszóval, hanem egyéb módon (pl. ujjlenyomattal) azonosítjuk magunkat. Igazából mindegy, hogy a nevünkre kattintva, a fényképünket kiválasztva vagy bármilyen egyéb módon „mutatkozunk be” a rendszernek, a háttérben az UID áll. Szintén a bejelentkezés során kapja meg a felhasználó a különböző, a rendszergazda által beállított csoporttagságokat, illetve az ehhez tartozó jogosultságokat. Ezeket az információkat lekérdezhetjük az id parancs segítségével. (Szenilis vagy tudathasadásos felhasználók azt is megkérdezhetik, hogy a rendszer őket milyen névvel látja, erre jó a whoami parancs, illetve a nagyon hasonló who am i is. Ez utóbbi a bejelentkezett felhasználók listáját lekérő who parancs kimenetéből csak a ránk vonatkozó információt jeleníti meg.)

Amikor bejelentkezett felhasználóként csinálunk valamit a gépen, a bejelentkezéskor (indirekt módon) kiválasztott UID határozza meg, hogy mit tehetünk a rendszerrel, ugyanis a jogosultságkezelés ezt (és esetleg a GID-et) ellenőrizve dönt, hogy valamit megtehetünk, vagy nem. Ha nincs

9. Egyéb UNIX, vagy UNIX-szerű rendszerben a 0-s csoportot szokták még system, vagy wheel névvel illetni

jogosultságunk a művelet elvégzésére, a következőket tehetjük:

– kijelentkezés után más névvel (tehát más UID-dal) visszajelentkezünk, nyilván olyannal, akinek van joga a művelet elvégzésére
– ideiglenesen azonosítót váltunk. Erre szolgál a su (superuser) parancs (illetve ha csoportjogra van szükségünk, a chgrp – change group – parancs). Ha mást nem mondunk, a su parancs rendszergazda (root) felhasználóvá vált, de lehetőségünk van egyéb felhasználói név megadásával más felhasználóvá „átváltozni”.
– ideiglenesen megszerezzük a szükséges jogosultságot. Ezt legtöbbször a sudo nevű paranccsal tesszük. A su/chgrp parancspárossal ellentétben a sudo egyszerre biztosítja a felhasználói, illetve a csoportjogosultságok megszerzését. Előbbit a -u, utóbbit a -g opcióval kérhetjük. A sudo hasonlóan a su-hoz, ha mást nem mondunk, akkor alapértelmezettként a root felhasználó jogait biztosítja.

Fenti három lehetőségből Linux alatt messze a sudo használata a legelterjedtebb. Annyira, hogy pl. az Ubuntu terjesztés alapból nem is teszi lehetővé a root felhasználó bejelentkezését, vagy a su parancson keresztüli root elérést.

A sudo használata

Amennyiben extra jogosultság kell egy művelet végrehajtásához, akkor a parancsot sudo parancs paraméterek formában kell begépelni, pl.:

sudo ip addr add 10.10.10.10/24 brd + dev eth0

Első alkalommal a sudo megkérdezi a saját felhasználói nevünkhöz tartozó jelszót[10], és ha jól gépeltük be, akkor extra, rendszergazdai jogokkal végrehajtja a paraméterként megadott parancssort. Egy időzítő mechanizmus segítségével biztosítja, hogy ha a következő sudo parancsot nem sokkal később gépeljük be, akkor ne kelljen újra jelszót gépelni. (A lejárati időt sokan indokolatlanul rövidnek tartják – általában 5-15 perc közötti értékre van beállítva; a rendszergazda ezt módosítani tudja a sudoers névre hallgató konfigurációs fájlban.) A sudo használata esetén a végrehajtható parancsok (vagy akár a parancsok és a megengedett paraméterek) listája felhasználónként, vagy csoportonként szabályozható. Precízen beállítani a sudo-t nem könnyű, de használatával az amúgy UNIX és Linux rendszereken nem létező operátor[11] felhasználói kategória megalkotható, és tényleg szeparálni lehet az egyes rendszergazdai jogosultságokat.

A sudo minden rajta keresztül végrehajtott műveletet naplóz, illetve ha valakinek nem sikerül az azonosítás, vagy nem megengedett parancsot próbál végrehajtani, akkor erről azonnal figyelmeztetőt küld a bekonfigurált módon (alapból e-mail a rendszergazdának). Ez eléggé eltér a su módszerétől, ahol a(z esetleg sikertelen) parancsvégrehajtás körülményei (ki, mit mikor stb.) tárolódnak egy szöveges fájlban, amit egyéb eszközzel nekünk kell figyelni.

Sok helyen a sudo konfiguráció nincs túlbonyolítva, és adott felhasználó, vagy felhasználói csoport minden tagja számára minden hozzáférést engedélyeznek. Mivel a sudo -i vagy sudo -s, esetleg a sudo su, netán a direkt sudo sh használata esetén már nem lesz nyoma a naplókban annak, hogy pontosan mi is történt, ezt ilyen konfiguráció használata esetén érdemes észben tartani.

10. A jelszókérés természetesen átkonfigurálható, vagy akár ki is kapcsolható
11. Itt most extra jogosultságokkal (pl. nyomtató alrendszer, hálózat stb. konfigurálása) rendelkező felhasználót (vagy csoportot) értünk alatta

Egy kis extra: SELinux

Egyre több Linux-terjesztésben érhető el a SELinux névre hallgató hozzáférés-szabályozási rendszer. Komplex, nagy tudású eszköz, konfigurálása nem könnyű. Amiért itt szerepel, az csak egy apróság: SELinuxot használó környezetben ugyan változatlanul léteznek a fent tárgyalt UID és GID fogalmak, de megjelent egy új, az ún. SELinux azonosító (SELinux identity). Mivel a SELinux hozzáférés-szabályozásában ez játszik szerepet, ezért érdemes róla tudni, illetve legalább annyit, hogy ha kell, lekérdezhető, hogy egy adott Linux-felhasználó milyen SELinux azonosítóval rendelkezik.

```
sudo semanage login -l
```

(Egyelőre ebben a könyvben nem tervezünk a SELinuxszal foglalkozni, komplexitása miatt ugyanis akár egy önálló könyvet is megérne.)

Fájlok, könyvtárak, dzsóker karakterek

A parancsértelmező egyik nagyon hasznos szolgáltatása, hogy a parancssorban a fájlok nevét minta segítségével is meg lehet adni. A minta megadásához használható karaktereket több különböző néven is emlegetik: helyettesítő karakterek, dzsóker karakterek, sőt az igen elborzasztó fájlnév metakarakter is feltűnik itt-ott a szakirodalomban. Angol nyelven leginkább a „wildcard pattern” kifejezéssel illetik. Van viszont egy olyan kifejezés, amit használni ugyan nem fogunk, de ismerni kell. Amikor a parancsértelmező az általunk begépelt parancssorban a lentebb tárgyalt karakterek helyettesítését elvégzi (azaz kicserél egy általunk adott mintát egy vagy több fájlnévre), az a művelet a GLOBBING. („Egyszer volt, hol nem volt, valamikor a pattintott kőkorszak idején, volt egy program, amit úgy hívtak: glob. Feladata az volt, hogy …” Akit a dolog mélyebben érdekel, man 7 glob.)

A három dzsóker karakter:

– ? A fájlnévmintában az adott helyen egyetlen tetszőleges karakter megadására használható. A rejtett fájlnevek nevének elején álló pont nem írható le mintával. (Más helyen álló pontok helyett írhatunk kérdőjelet.) Ezen kívül is van kivétel: ez a karakter nem helyettesíti a fájlnévben a 0 kódú karaktert (C nyelvű megfogalmazásban: a ’\0’ karaktert) – némileg enyhítheti a gyötrő hiányt, hogy elvi szinten ilyen karakter nem lehet fájlnév része. Hasonlóan nem helyettesíthető a fájlnévben szereplő törtvonal. Hozzáértők (vagy akik figyelmesen előreolvasták a fájlkezelés részt és emlékeznek, hogy volt róla szó) természetesen már tudják, hogy a / sem lehet része a fájl nevének – csak az elérési úttal adott névben szerepelhet /. Mindenesetre egy másik, kicsit gyakorlatiasabb megfogalmazás: elérési úttal adott fájlnév esetén az egyes komponenseket elválasztó / karakter nem helyettesíthető dzsóker karakterrel. Nem csak a ?, de a későbbiekben taglalt karakterek egyike sem alkalmas a / helyettesítésére.
– [...] A fájlnévmintában az adott helyen egyetlen karakter megadására használható, hasonlóan a ?-hez. Lényeges különbség, hogy míg a ? bármilyen karakter helyettesítésére alkalmas, a [...] forma használatakor csak a zárójelek között felsorolt karakterek szerepelhetnek a fájlnév adott helyén. Szintén nem lehet vele megadni a név elején álló pontot.
– [!...] Majdnem ugyanaz, mint az előző, csak éppen nem a felsorolt karaktereket, hanem azokon kívül minden egyéb karaktert jelent – persze a rejtettség jelzésére szolgáló név eleji pontot, vagy az elérési útvonal jelzésére használt törtvonalat így sem lehet mintával megadni.
– * Bármilyen karaktersorozatot jelenthet. Hivatalosan: tetszőleges darabszámú tetszőleges karakter – kivéve a név eleji pontot, meg az elérési útvonal /-ját.

Figyelmesebb olvasónak feltűnhetett, hogy voltak olyan kikötések, amelyek mindegyik globkarakter esetén szerepeltek:

– A rejtett fájlnevek elején álló pont – ellentétben a fájlnév bármely egyéb pontjával – nem írható le mintával.
– Akár abszolút, akár relatív elérési útvonallal adunk meg egy vagy több fájlnevet, az alkönyvtárak jelzésére szolgáló törtvonal (/) nem írható le mintával. Elérési útvonal használatakor tehát az egyes komponenseket adhatom meg mintával, de a /-t ki kell írni. (Azaz a*b, vagy akár a?b nem jelenti az a/b fájlt. Cserébe */*-ként már hivatkozhatunk rá – igaz, ha van másik, nem rejtett nevű alkönyvtár, és abban nem rejtett nevű fájl, azt mindet jelenti ez a minta.)

Nem hátrány tudatosítani magunkban, hogy a parancsértelmező a fájlnevek helyettesítését

(mint egyébként az egyéb helyettesítéseket is), az általunk begépelt parancs elindítása előtt hajtja végre – következésképpen az elindított parancs már nem fog róla tudni, hogy az eredeti parancssorban szerepeltek-e helyettesítő karakterek – a parancs már csak néhány fájlnevet lát. További jól-jöhet-még információ, hogy a helyettesítés során a parancsértelmező a nyelvi beállításokban meghatározott rendezési elvet követő ábécésorrendben adja át a helyettesített fájlneveket. Jó tudni azt is, hogy az egyes shellek beállítástól függő módon reagálnak arra, ha mi megadunk egy mintát, de a mintára illeszkedő fájlnév nem létezik. Ilyenkor akár hibaüzenetet is kaphatunk a parancsértelmezőtől (és mellesleg ekkor nem hajtja végre a parancsot). Más parancsértelmezők, más beállítás esetén (ez a gyakoribb egyébként) mindenféle probléma nélkül odaadják a mintát az alkalmazásnak, csináljon vele, amit akar. Ez utóbbi működésnek furcsa következménye is lehet. Például ha egy nem üres könyvtárban kiadjuk a vi *.txt parancsot, akkor ha ott van néhány fájl, amelynek illeszkedő neve van, akkor a vi szövegszerkesztő ezeket kezdi szerkeszteni. De ha a könyvtár üres (vagy csak nincs benne txt kiterjesztésű fájlnév), akkor a vi megkapja ezt a paramétert, amit aztán a frissen létrehozandó nevének tekint, tehát ha írunk valamit a fájlba, majd mentjük, akkor a kedves nevű „*.txt”-t hozzuk vele létre.

A követező példákban szokásunktól eltérően megint jelezzük a promptot. (És kivételesen tekintsünk el az ékezetes fájlnevektől. Ez itt most példa, nem éles környezet.):

```
$ ls -A[12]
.dió alma banán eper körte
narancs szilva
$ ls -d ????
alma eper
$ ls -d ??[mn]*
alma banán
$ ls -d [!abe]*
körte narancs szilva
$ ls -d *a
alma szilva
$ ls -d *r*
eper körte narancs
$ ls /ho*/*
/home/less /home/tanf1 /home/tanf2 /home/tanf3
```

12. Ha valaki nem értené kapásból. Az ls -A opciója ugyanúgy kiírja a rejtett fájlokat, mint az ismertebb -a, csak épp a listából kihagyja a . és a .. bejegyzéseket. Fenti parancs amúgy kiírja az aktuális könyvtárban levő fájlok nevét.

Fájlkezelő parancsok

Fájlnevek

Linuxon az adatokat fájlnak nevezett struktúrákban tárolják. Az adatoknak nevük van, ezt használjuk az adat azonosítására, elérésére – ez a fájlnév. Ami viszont elsőre elég szokatlan, hogy ugyanahhoz a fájlhoz nem csak egy, hanem több különböző *valódi fájlnév,* ún. *hard link* is tartozhat. Ezek a fájlnevek egyenrangúak, és csak az utolsó törlése esetén törlődik a fájl tartalma is a fájlrendszerből. Linux alatt a fájlok neve tetszőleges karaktereket tartalmazhat – és ebbe olyan rondaságok is beletartoznak, mint szóköz, soremelés, vagy három darab pluszjel. Ráadásul a kis- és nagybetűket megkülönbözteti. Noha elvi síkon nem okoz problémát ékezetes karakterek használata a fájlnevekben sem, időről időre találkozunk hibás, csak az angol nyelv ékezettelen betűire felkészült programokkal, ezért ez ügyben óvatosságra intünk mindenkit, kifejezetten nem javasoljuk a használatát[13]. (Fájl nevében nem megengedett a C programozási nyelvben '\0' jelöléssel illetett, azaz a 0 kódú karakter, illetve mivel speciális szerepe van, ezért szintén nem lehet a fájl nevének része a /. Az más kérdés, hogy – a később tárgyalandó – elérési útvonallal adott név már tartalmazhat /-t.)

Noha pont is szerepelhet egy fájlnévben (akár több is), ennek ellenére van egy specialitása: azokat a fájlokat, amelyek neve ponttal kezdődik, a rendszer a többi fájltól kicsit eltérően kezeli, elrejti[14]. Az ilyen fájlokat „rejtett fájl"-nak nevezik. (Elsősorban programok beállításait tartalmazó fájlok, könyvtárak szoktak rejtettek lenni.) A fájlnévben nem különböztetjük meg a „kiterjesztést", de köznapi beszédben általában szokták érteni, ha valaki ezt a kifejezést használja[15].

Fájltípusok

A „fájl" egy gyűjtőfogalom, különböző fájltípusok léteznek. A legfontosabbak:

– közönséges fájl (ordinary file) – egyéb operációs rendszert használók ezt szokták fájlnak nevezni; szöveget, képet, hangot, futtatható program kódját (és í. t.) tároló struktúra. A közönséges fájl jele az ls parancs kimenetében: - (egy mínuszjel).
– könyvtár (directory) – egyéb rendszerben pl. mappa (folder) névvel illetik, szerepe szerint az egyéb fájlok tárolására szolgál, segítségével ésszerű, logikus, fastruktúrájú hierarchiát építhetünk ki az adattároláshoz. Jele: d.

13. Pl. legutoljára az eléggé elterjedt ZIP formátumban futottunk bele abba, hogy egy meg nem nevezett nem szabad operációs rendszer alatt ZIP-be csomagolt német nyelvű fájlnevek eléggé sajátosan néztek ki Linuxon kicsomagolás után (és még van jó pár ilyen jellegű rossz tapasztalatunk).
14. Igazság szerint csak a parancsértelmező, az ls parancs és a fájlkezelő programok rejtik el ezeket alapértelmezés szerint, de ez bőven elég.
15. Vajon mi a kiterjesztése az alma.körte.szilva nevű fájlnak? (Hallgatólagos megállapodás alapján általában az utolsó pont utáni részt tekintjük annak – de még egyszer hangsúlyozzuk, az operációs rendszer szintjén a fájlnévkiterjesztés, mint fogalom, nem létezik.)

– szimbolikus link (szimbolikus fájlnév, symlink, vagy soft link néven is emlegetik)[16] – olyan fájltípus, amely egy másik fájl nevére hivatkozik[17]. Jellemző használat pl., amikor egy nagyon hosszú elérési útvonallal elérhető fájlra hivatkoznak symlinken keresztül.[18] Jele: l.

Ezen a három fájltípuson kívül még négyféle létezik. Rendszergazdák számára fontos a következő kettő:

– karakteres (vagy karakter speciális, character special) eszközfájl. Jele: c.
– blokkos (blokk speciális, block special) eszközfájl. Jele: b.

Nevükből sejthető, ezek a különböző hardverkomponensek, perifériák elérésére szolgálnak. Karakteres fájlok esetén azt feltételezzük, hogy a hardverrel egyszerű, „byte-okat írok-byte-okat olvasok” módon lehet kommunikálni, míg blokkos esetben az operációs rendszer puffereli az adatokat, egyszerre nagyobb mennyiségű adatot küld, illetve fogad az adott eszközre/ről.

Az utolsó két fájltípusról nem árt tudni, de ritkán szoktak ilyeneket direktben piszkálni:

– cső (pipe) – Két, ugyanazon a gépen futó program közötti egyirányú adatcserére használt objektumról van szó. Két alfaja a névtelen (no-name, vagy unnamed pipe) és a névvel ellátott cső (named pipe, FIFO). Nyilván csak az utóbbi látható valahol a könyvtárstruktúrában, azaz fájlként. Jele: p.
– socket (foglalat) – ez a csőhöz nagyon hasonló objektum, csak éppen kétirányú kommunikáció folyhat egy socketen keresztül. Ebből is kétféle létezik. A régebben Unix domain, mostanság local domain néven emlegetett – mondjuk helyi. Ennek a fajtának a használatához a két programnak ugyanazon a gépen kell futnia. Ez a fajta az, amelyik látszik a könyvtárstruktúrában. A második az ún. internet domain típusú (mondjuk úgy: távoli), ennek használatakor a két egymással kommunikáló programnak nem kell azonos gépeken futni, elég, ha hálózaton keresztül látják egymást. Jele s.

Elérési útvonalak

A különböző fájlok a fentiek alapján könyvtárakban tárolódnak, a könyvtárakban alkönyvtárak, abban al-alkönyvtárak, és így tovább. A struktúra tetején álló könyvtár neve gyökérkönyvtár (root directory), és a / névvel nevezzük. Minden könyvtár tartalmaz két speciális nevű fájlt: . (aktuális könyvtár) és .. (szülőkönyvtár). Angol nyelvű dokumentációban sokszor dot és dotdot néven szerepelnek. A . mindig azt a könyvtárat jelenti, amelyikben szerepel, a .. pedig a struktúrában eggyel fölötte álló könyvtárat. A gyökérkönyvtárban levő fájlok[19] neve pl. /tmp, /root, /vmlinuz. A /.. kicsit speciális, hisz a gyökérkönyvtár fölött nincs másik, így az szintén a /-t jelenti. Egy, a gyökérkönyvtár közvetlen alkönyvtárában levő fájl neve pl. /etc/passwd vagy /tmp/.X11 – és ezt lehet folytatni, pl. /usr/share/doc/zfs/README. Ez az ún. *abszolút elérési úttal adott fájlnév*, röviden *abszolút név*[20]. Az abszolút név egyértelműen arról ismerhető fel, hogy a root könyvtár nevével, azaz /-rel kezdődik. Létezik egy ún. *relatív fájlnév* is, ebben az esetben az aktuális könyvtártól kezdve

16. egyike azoknak az elég fontos alapfogalmaknak, amelyekhez még nem találtunk frappáns magyar elnevezést. Még a legjobbnak tekinthető „gyenge kötés/gyenge kapocs” is borzasztó.
17. azért nem „másik fájlra hivatkozik”, mert előfordulhat, hogy a symlink által hivatkozott fájl nem is létezik (esetleg ideiglenesen nem elérhető)
18. noha elsőre eléggé hasonlítanak, a hard link és a soft link két nagyon különböző dolog
19. fájl, mint az összes fájltípus összessége, nem pedig mint közönséges fájl
20. Ilyenkor a gyökérkönyvtártól kezdve leírjuk, hogy a struktúra tetejétől indulva, milyen könyvtárakon keresztül jutunk el az adott fájlig.

nézzük az elérési útvonalat. Így jön létre a bin/ls (az aktuális könyvtár bin nevű alkönyvtárában levő ls nevű fájl) vagy a ../../lib/termcap (az aktuális könyvtár szülő könyvtárának szülő könyvtárában levő lib alkönyvtárban levő termcap nevű fájl) formájú név. Relatív név sosem kezdődik /-lal. (Mind abszolút, mind relatív név esetén szerepelhet . vagy .. az útvonalban, de jellemzően inkább relatív névben szokták használni.)

Könyvtárak kezelése

Bejelentkezés során mindig bekerülünk valamilyen könyvtárba. Ez az ún. *sajátkönyvtár* (HOME directory). A könyvtárstruktúrában adott pillanatban elfoglalt helyünk neve pedig *munkakönyvtár* (working directory), vagy *aktuális könyvtár* (current directory). (Bejelentkezéskor természetesen a HOME lesz a munkakönyvtár.) A munkakönyvtár váltására szolgál a cd (change directory) nevű parancs:

```
cd /tmp
```

Ha a cd parancsot paraméterek nélkül indítjuk el, visszakerülünk a HOME-ba. Aktuális helyünk megjelenítésére pedig a pwd (print working directory) szolgál.

```
pwd
```

Ahol jogosultságunk van hozzá, ott módosíthatjuk a könyvtárszerkezetet is. Új könyvtár létrehozására szolgál az mkdir (make directory) parancs, egy üres könyvtár pedig törölhető az rmdir (remove directory) paranccsal.

```
mkdir alma alma/körte
rmdir alma/körte alma
```

Nem-könyvtár típusú fájl, vagy nem üres könyvtár törlésére az rmdir nem használható.

Alapvető fájlkezelő parancsok

Egy közönséges fájl tartalmának megvizsgálására a cat parancs használható:

```
cat fájl1.txt
cat fájl1.txt fájl2.txt fájl3.txt
```

Mivel lehet, hogy a fájl túl hosszú, és a szöveg kiszalad a képernyőről, ezért speciális esetben használhatóak a head és a tail parancsok. Előző a paraméterként megadott számú (alapértelmezés szerint 10) sort írja ki a fájl elejéről, utóbbi ugyanígy, csak a fájl végéről.

```
head -8 /etc/passwd
tail -n 3 /etc/group
```

(Érdemes utánanézni a tail -f opciójának, illetve annak, hogy mit jelent, ha pozitív előjellel adjuk meg a sorok számát.) Ha azonban tényleg a teljes fájl tartalmát szeretnénk megjeleníteni, akkor érdemesebb valamely lapozóprogramot használni. Akik Unix környezetből jöttek, azok a more, akik eleve Linuxon szocializálódtak, azok viszont általában inkább a sokkal többet tudó less programot használják.

```
more /root/.profile
less /etc/profile
```

A cat, head, tail, more, less parancsok majdnem minden fájltípus esetén használhatók a fájl tartalmának megjelentésére, de könyvtárak esetén nem. Fent szerepelt, hogy a könyvtárak „tartalma” az egyéb fájlok (neve és néhány jellemzője). Könyvtárak listázására szolgál az ls nevű parancs. Paraméterek nélkül csak az aktuális könyvtárban (azaz a .-ban) szereplő fájlok nevét írja ki ábécérendben – de a rejtett fájlok a listában nem szerepelnek. Adható az ls-nek könyvtárnév paraméter, akkor azt listázza az aktuális könyvtár helyett. Az ls az egyik legtöbb opcióval rendelkező linuxos parancs, érdemes utánanézni. Fontosabb opciói:

- -a all, azaz az összes fájl szerepel a listában, az ún. rejtett fájlok is
- -l hosszú (long) lista, azaz a néven kívül a fájl egyéb jellemzői (tulajdonos, méret, módosítási dátum, stb) is szerepelnek a listában
- -t módosítási dátum (és nem ábécérend) szerint rendezett lesz a lista
- -r a rendezést megfordítjuk
- -d nem a könyvtár tartalmát, hanem magának a könyvtárnak (az egyéb paraméterek segítségével meghatározott) jellemzőit listázza ki

Egy példa ls -la :

```
$ ls -la
total 202516
drwxr-xr-x  4 user staff  16384 dec   1 17:07 ./
drwx--x--x 87 user staff  36864 dec   1 16:00 ../
-rw-rw-r--  1 user staff  873999 nov   26 22:18 Alapfok_B1.mp3
-rw-rw-r--  1 user staff  616820 szept 27 15:50 asterisk-gui-2.1.0-rc1.tar.gz
drwxrwxr-x  9 user staff  4096 aug    1 14:56 Asterisk-hun/
-rw-------  1 user staff  7708280 okt   29 10:42 documents-2013-10-29.zip
-rw-rw-r--  1 user staff  1225947 nov   26 22:20 Drew_Barrymore_Is.mp3
-rw-rw-r--  1 user staff  48659368 nov   14 11:42 google-chrome-stable.deb
-rw-rw-r--  1 user staff  23049984 aug   25 08:53 Indiai képeskönyv_1.pdf
-rw-rw-r--  1 user staff  59656 nov   6 12:21 Kiosztas-v3.eml
-rw-rw-r--  1 user staff  136957 júl   10 13:46 novacom-linux-64.tgz
-rw-rw-r--  1 user staff  51387 febr  18  2013 oklevél.odt
-rw-rw-r--  2 user staff  8126464 szept  4 11:22 openwrt.bin
-rw-rw-r--  1 user staff  1933179 nov   26 22:19 Puppy_Love.mp3
-rw-rw-r--  1 user staff  2405 aug   29 09:36 rules.txt.gpg
-rw-rw-r--  1 user staff  2742993 nov   26 22:19 Ryanair_.mp3
-rw-rw-r--  1 user staff  4265 júl   12 15:22 slideshow.txt
```

Az egyes sorokban a következő információk láthatók: az első oszlopban a fájlok típusát[21] és alap UNIX-jogosultságait[22] látjuk, utána a fájlok hard linkjeinek számát, ezt követi sorban a fájl tulajdonosa, csoporttulajdonosa, a mérete, a módosítási dátuma, végül a neve.

Másolás, átnevezés, ...

Egy létező fájlról másolat a cp (copy) parancs segítségével hozható létre. Ha átnevezni, vagy a könyvtárszerkezetben átmozgatni szeretnénk egy fájlt, akkor arra a mv (move) parancs szolgál. És egy létező fájlhoz link (akár hard, akár soft) az ln (link) paranccsal hozható létre. A parancsok nagyon hasonló szintaxissal használhatók:

```
cp fájl1.txt fájl2.txt
```

21. a fejezet elején levő felsorolásban szerepelt, hogy mely fájltípust milyen karakter jelöl az ls parancs kimenetében
22. a jogosultságok értelmezése egy másik fejezetben található

```
mv fájl1.txt /tmp/fájl3.txt
ln /tmp/fájl3.txt fájl2.txt /var/tmp
ln -s fájl2.txt fájl4.txt
```

Az első paranccsal másolatot készítünk az aktuális könyvtár fájl1.txt nevű fájljáról, ugyanide, fájl2.txt néven. A következő parancsban ugyanezt a fájlt átmozgatjuk a /tmp könyvtárba és ugyanakkor új nevet is adunk neki. Ezt követően az átmozgatott fájlhoz, és az első lépésben készített másolathoz is csinálunk egy-egy hard linket a /var/tmp könyvtárban[23]. Mivel nem adtunk meg mást, így az „eredeti" és a link fájlok ugyanolyan nevűek lesznek. Végül az utolsó parancs segítségével csináltunk egy szimbolikus linket fájl4.txt néven az első lépésben másolással létrehozott fájl2.txt-hez.

Fájlok törlése

Fájlok törlésére az rm parancs használható. Hasonlóan az rmdir-hez, háklis a törlendőre, ez a parancs könyvtárakat nem hajlandó (alapértelmezetten) törölni. Legfontosabb opciói a „-i" – ekkor minden törlendő fájlra egyesével rákérdez; a „-f" – ez pont az ellentéte, ekkor nincs semmiféle kérdés, figyelmeztetés, töröl; végül a „-r", aminek hatására végre hajlandó könyvtárakat is törölni – ilyenkor az egész törésre megadott könyvtárfát kitörli.[24]

```
rm fájl1.txt
rm -r -i alkönyvtár
```

A második parancs a teljes törlendő könyvtárfa minden egyes bejegyzésére egyesével rákérdez (és ha akár csak egyet is nem hagytunk törölni, a könyvtár maga megmarad – meg persze aminek elutasítottuk a törlését).

Fájlok tömörítése

Egyes fájlokat tömöríthetünk, erre jó pl. a gzip nevű tömörítő (megfelelő opcióval híva, a kicsomagolásra is jó).

```
gzip próba.txt
```

Eredményeként a próba.txt bekerül a próba.txt.gz nevű tömörített fájlba, majd az eredeti fájl törlődik!

```
gzip -d próba.txt.gz
```

Ez kitömöríti az előző fájlt (és a tömörített verzió törlődik).

Ha viszont több fájlt szeretnénk egybecsomagolni, általában a tar nevű eszközt használják. Ez alapértelmezetten nem tömörít, de megfelelő opcióval indítva, fenti gzip[25] segítségével a végeredményként előálló archívumot betömöríti.

```
tar cf kimenet.tar ezt.txt meg-azt.txt meg-amazt.txt
```

A paraméterekben szereplő három darab szöveges fájlt csomagolja egybe, a kimenet.tar fájlba.

```
tar tvf kimenet.tar
```

Kilistázza az archívumban szereplő fájlok nevét (és a „v" opció megadása miatt a fájlok egyéb jel-

23. ez a művelet csak abban az esetben fog sikerülni, ha a forrás és a célfájlok ugyanabban a fájlrendszerben vannak (lásd a háttértárakról szóló fejezetet)
24. klasszikus beavatásnak számít kezdő rendszergazdát rábeszélni, hogy adja ki az „rm -rf /" parancsot
25. persze más tömörítőt is beállíthatunk

lemzőit is):

```
tar xf kimenet.tar meg-amazt.txt
```

Az archívumból egyedül a „meg-amazt.txt” nevű fájlt csomagolja ki.

Elsősorban különböző rendszerek közötti adatcsere esetén lehet érdekes, hogy egyéb operációs rendszerek elterjedtebb tömörítőprogramjai általában Linux alatt is elérhetők (ritkábban csak a kitömörítő): a ZIP, ARJ, RAR fájlok kezelhetőek.

Fájlkeresés

Sűrűn előforduló igény valamilyen fájljellemző alapján megkeresni a fájlokat. Erre szolgál a find parancs. Most szólunk, a find önmagában nem képes a fájl tartalma alapján keresni – arra a grep família parancsai szolgálnak. (Részletesebb leírás a „Szabályos kifejezések” c. részben.) A find-dal tehát tartalmon kívül egyéb jellemzők alapján szokás keresni:

– tulajdonos
– csoporttulajdonos
– vagy pont ellenkezőleg, ismeretlen tulajhoz/csoporthoz tartozó fájlok
– méret alapján
– módosítási dátum alapján
– stb.

A hagyományos Unix rendszerekben levő find kétféle paramétert vár:

– hol keressen
– milyen feltételek alapján keressen

A Linuxokon elterjedten használt find verzió annyiban tér el, hogy az első elhagyható. Ha tehát nem adjuk meg a keresés helyét, akkor az aktuális könyvtárban indul a keresés, és ebben, és ennek alkönyvtárában keres. Ha máshol akarunk kerestetni, akkor egyszerűen adjuk meg annak (azoknak) a könyvtárnak a nevét, ahol keresni szeretnénk.

A feltételek általában egy kötőjellel kezdődő rövid kifejezésből és legtöbbször a hozzá tartozó paraméterből állnak. A fenti lista immár find-nyelven:

– -user team5
– -group staff
– -nouser / -nogroup
– -size +10M
– -mtime -7

Amennyiben több feltételt adunk meg, minden feltételnek teljesülnie kell. (Ha nem ezt szeretnénk, akkor zárójelekkel, és speciális logikai operátorokkal írhatjuk elő az igényeinket.) Példa:

```
find / -nouser -o -nogroup
```

Minden olyan fájl kilistáz, amelynek ismeretlen tulajdonosa, vagy ismeretlen csoporttulajdonosa

van.[26]

```
find /home -atime +365 -size +1M
```

Az atime az ún. *access time* – az utolsó olvasás ideje. Azaz a felhasználók könyvtárában keressük azokat a fájlokat, amelyekhez egy éve nem nyúlt senki, és 1 MB-nál nagyobbak. A megtalált fájlok nevét a find alapból kiírja a szabályos kimenetre, de a -exec vagy a -ok segítségével tetszőleges parancsot is indíthatunk.

```
find . \( -name \*.bak -o -name \#\* \) -exec mv {} /kuka \;
```

Fenti parancsban a sok rep-jel azért kell, mert a shell elől el kell takarni az ott található speciális karakterek jelentését. A zárójelekkel a kétféle névkeresést fogtuk össze: akár .bak-ra végződik egy fájl neve, akár #-karakterrel kezdődik, a műveletet végre szeretnénk hajtani. A végrehajtandó parancs pedig a megtalált fájlokat egyesével átmozgatja a /kuka nevű könyvtárba. Sajnos a példa több sebből vérzik.[27] Képzeljük el, hogy nem a különböző szövegszerkesztők által létrehozott mentési fájlokat teszem ezzel a paranccsal „kukába”, hanem az általam fontosnak minősített adatokról mentést szeretnék készíteni. Fontos dolgaimat minden esetben Fontos-* néven nevezem el, és készítek hozzá egy *.md5 ellenőrzőfájlt is. És természetesen nem átmozgatom, hanem másolatot készítek egy erre a célra fenntartott RAID-tömbre, amit a /backup alá csatolok fel. Azaz a parancs erre változik:

```
find . \( -name \*.md5 -o -name Fontos-\* \) -exec cp {} /backup \;
```

A lábjegyzetben szereplők miatt ez sajnos minden, csak nem biztonsági mentés. Aki szerint ez így jó, az inkább olvassa el a Mentés fejezetet.

26. Azaz a fájl metaadataiban (inode) szereplő UID- vagy GID-értékhez nem tartozik felhasználónév / csoportnév a rendszer adatbázisában
27. Legalább kétféle elég súlyos logikai hiba van benne (és egy, a teljesítmény szempontjából nem szerencsés megoldás), ezért kérjük, ne próbálkozz meg ezt élesben használni! Megvannak a hibák? Ha nem, olvasd el a következő oldalon a magyarázatot!

1. hiba: mivel a mv és a cp (sőt az ln is) működés szerint

```
cp ./a/b/c/d /backup
```

/backup/d nevű fájlt hoz létre (amennyiben a „/backup” létező könyvtár neve), ebből következik, hogy ha egy későbbi könyvtár esetén

```
cp ./x/y/d /backup
```

lenne a végrehajtandó parancs, ezzel az utóbbi fájllal már felülírnánk az első lépésben „mentési” céllal odamásolt fájlt.

2. hiba: Ha valami szórakozott pillanatunkban ezt a / könyvtárban állva adnánk ki, akkor amikor a find épp a /backup könyvtárban jár, akkor a cp egy fájlt önmagára szeretne másolni, ami nem biztos, hogy jó ötlet. A Linuxokban levő cp észreveszi a hibát, tehát nem lesz adatvesztés, de ettől függetlenül ekkora logikai hibát nem jó véteni.

3. hiba: Ez valójában csak erőforrás-pazarlás, de mindegyik fájlhoz új példány cp (mv) parancs indul, holott ezek több fájlnevet is elfogadnak. Ezen hiba kiküszöbölésére szokták az xargs nevű parancsot javasolni „find … | xargs cp” formában, de az általános esettől eltekintve ezt épp a find is tudja:

```
find . \( -name \*.md5 -o -name Fontos-\* \) -exec cp {} /backup +
```

A változás a find -exec paraméterének végén látható, ahol a \; helyett + jel áll. Jelentése: annyi paramétert gyűjtsön egybe, amennyit csak tud, és nem minden fájlra egyesével, hanem sok paramétert összegyűjtve futtassa le a parancsot.

Jogosultságok, azok használata

Klasszikus UNIX-szerű jogosultságok

A Linux rendszerek alapszintű jogosultságkezelése a UNIX-ból származik, és meglehetősen egyszerű. Felhasználók és fájlok egymáshoz való (tulajdon)viszonyán alapul. Minden fájlnak van egyetlen tulajdonosa (ezt egy UID reprezentálja a rendszerben), és egyetlen ún. csoporttulajdonosa (ezt pedig egy GID). Ezek a jellemzők a fájl létrehozásának pillanatában állítódnak be, de utólag módosíthatók (chown, chgrp parancsok.) Ezek az információk az „ls -l" paranccsal lekérdezhetők.

Minden felhasználó minden bejelentkezése során automatikusan megkapja a hozzá kiosztott egyetlen felhasználói azonosítót (UID), és a neki kiosztott egy vagy több csoportazonosítót (GID-eket) a rendszertől. (Ezeket a rendszergazda állítja be akkor, mikor a felhasználót beilleszti a rendszerbe. Utólag is neki van joga ezeket változtatni.) A felhasználó bizonyos feltételek teljesülése esetén, ideiglenes jelleggel megváltoztathatja a felhasználói azonosítóját (su, sudo parancsok) illetve a csoporttagságát (newgrp parancs)[28]. A felhasználó ID információi pedig az „id" nevű parancscsal kaphatók meg.

Végül pedig amikor valaki (valami) elindít egy programot, a futó program, azaz *folyamat* (process) örökli az elindítója jogait – technikailag a felhasználói és csoportazonosítóját[29] (azaz az UID és GID információkat). Ez szintén lekérdezhető, pl. a „ps -eo user,group,cmd" paranccsal.

A felhasználók tevékenységük során jellemzően fájlokhoz nyúlkálnak. Például amikor valaki kiadja a cat /etc/passwd parancsot, akkor első lépésként meg kell keresni a cat parancsot tartalmazó fájlt a PATH környezeti változó által meghatározott könyvtárakban (ehhez ezekhez a könyvtár típusú fájlokhoz kell keresési jog), utána programként el kell indítani a /bin/cat fájlban található kódot (ehhez a fájlra kell futtatási jog), ha ez sikerült, akkor a cat folyamatnak meg kell keresnie a /etc/passwd nevű fájlt (ennél a pontnál megint keresési jogok szükségesek a különböző könyvtárakhoz), végül olvasásra meg kell nyitni a fájlt (amihez szintén jog – most épp olvasási – kell). Már csak az a lépés van hátra, hogy a beolvasott adatokat meg kell tudni jeleníteni a kijelzőn – ehhez persze a kijelzőt reprezentáló fájlhoz kell valamilyen – nyilván írási – jog.

Fenti példákból szépen körvonalazódik a Linuxokon létező jogosultságok köre:

– olvasási jog, jele r (read)
– írási jog, jele w (write)
– futtatási jog, más néven keresési jog, jele x (execute)

A felhasználókat 3 kategóriába soroljuk.

– Van a fájl mindenkori tulaja (jele u, mint user)

28. Fentiek miatt megkülönböztetünk valós UID-ot (real UID, RUID) és hatásos UID-ot (effective UID, EUID). Az első az, amit a bejelentkezéskor oszt ki a rendszer, a második, amire változik a sudo/su hatására. Hasonlóan létezik RGID és EGID is.
29. És rögtön el is érkeztünk az első kivételig. Az ún. setuid vagy setgid beállítások esetén a létrejövő folyamat nem az őt indító felhasználótól, hanem a programkódot tartalmazó fájltól örökli a jogosultságokat (illetve az UID/GID értékeket).

– Vannak a fájl csoportjába tartozó felhasználók (g, mint group)
– És van az összes többi felhasználó (o, mint other)

Külön van meghatározva a tulaj joga, ettől teljesen függetlenül a csoporthoz tartozók joga, és a „maradék” felhasználók joga. Az ellenőrzés is ebben a sorrendben történik:

– Elsőként azt ellenőrzi a rendszer, hogy a fájl tulaja akar-e csinálni valamit. Ha igen, akkor a tulaj jogai határozzák meg *egyedül*, hogy a művelet elvégezhető-e. (Ebben az esetben a csoport és egyéb kategóriák jogai nem számítanak, azt a rendszer már nem is vizsgálja.)
– Ha nem a tulaj tevékenykedik, másodikként azt vizsgálja a rendszer, hogy csoporttag akar-e valamit csinálni. Ha igen, akkor a csoportjogok döntenek *egyedül*. (Azaz már nem vizsgálja a rendszer az egyéb felhasználók jogait.)
– Ha a művelet végrehajtója nem a fájl tulaja, és nem is tagja a fájl csoportjának, akkor az egyéb felhasználókra vonatkozó jogosultságok döntenek.

Fentiből az a kicsit furcsa helyzet is következik, hogy akár az is előállhat, hogy a csoporttagoknak vagy épp az egyéb kategóriának több joga van, mint a tulajnak.

Az egyes jogosultságok az egyes fájltípusokon némiképp eltérően működnek. Az olvasás és az írás jogok a legtöbb fájltípusnál egyértelműek, általában az adott típusú fájlban elérhető adatok elérhetőségét vagy módosíthatóságát jelentik. A futtatási/keresési jog csak közönséges és könyvtár típusú fájlokon jelent egyáltalán valamit.

jogok ---------- fájltípusok	olvasás (read)	írás (write)	futtatás(execute) / keresés(search)
közönséges fájl (-)	amit a neve mond	amit a neve mond	programként elindítható
könyvtár (d)	megnézhető a tartalma (ls, de -l opció már nem); a benne levő fájlokhoz csak ezzel a joggal nem férhetünk hozzá	létrehozható, átnevezhető, törölhető (!!!) benne fájl	be lehet lépni (cd), ismert nevű fájl jellemzője lekérdezhető (ls -l fnév), ismert nevű fájlhoz hozzá lehet férni (ha a fájl saját joga engedi)
symlink (s)	---	---	---
karakteres eszköz (c)	amit a neve mond	amit a neve mond	---
blokkos eszköz (b)	amit a neve mond	amit a neve mond	---
cső (p)	amit a neve mond	amit a neve mond	---
socket (s)	amit a neve mond	amit a neve mond	---

A táblázatban a „---” azt jelenti, hogy azokat a jogokat a rendszer nem értelmezi[30]. Fentiekből talán a könyvtárak jogainak értelmezése a legnehezebb. Nézzük kicsit részletesebben:

Ha egy könyvtáron csak olvasási jog szerepel (és a keresési jog nem), akkor ls-sel listázható a benne levő fájlok neve. De más információ a fájlokról nem kapható meg, azaz az „ls -l” már hibát ad vissza, vagy nem használható olyan ls opció (pl. „-t”, dátum szerinti rendezés), amely ezen egyéb információkat használná fel. Ami a kellemetlenebb, ha csak olvasási jogunk van egy könyvtárra, a fájlok nevét látjuk (de már típusát nem), és nem is férhetünk hozzá a fájlhoz (függetlenül az általunk egyébként nem látható fájlengedélyektől). Azaz akár az is lehet, hogy magához a fájlhoz lenne jogunk hozzáférni, de a fájlbejegyzést tartalmazó könyvtár joga megtagadja ezt a hozzáférést.

A könyvtár keresési joga pedig azt teszi lehetővé (azaz ha csak az x jog adott, és az r jog nem adott), hogy az ismert nevű fájlokról[31] megkaphassam a bővebb információkat („ls -l”); vagy épp használhassam ezeket a bővebb információkat („ls -t”); vagy ezen információk felhasználásával megpróbálhassam elérni a fájlokat (futtatni, olvasni, írni) – amihez viszont már magának a fájlnak a saját joga szintén szükséges. Ezen kívül könyvtár keresési joga teszi lehetővé az adott könyvtárba belépést (azaz kiadható a cd/dir parancs, és még sikerülhet is).

Könyvtár írási joga pedig biztosítja a könyvtár módosíthatóságát, amibe sokak számára igen váratlan és fájdalmas módon azt is jelenti, hogy egy fájl törlése nem a fájl saját jogain, hanem a fájlbejegyzést tartalmazó könyvtár írási jogán múlik (amúgy a törléshez a könyvtár keresési joga is szükséges).

chmod

A jogosultságok beállítása a fájl mindenkori tulajdonosának hatásköre (illetve a rendszergazda – root felhasználó – szintén módosíthatja azokat). A parancs használata egyszerű:

```
chmod JOGOK fájl1 fájl2 ...
```

Két módszer létezik a jogok megadására:

– numerikusan
– szimbolikusan

írhatjuk le a jogokat.

A numerikus forma a nehezebb. Minden jog egy számérték:

– r = 4
– w = 2
– x = 1

Meghatározzuk, hogy az egyes felhasználói kategóriák (tulaj, csoport, egyéb), pontosan milyen jogokat kapjon. A megkapandó jogok számértékét minden kategóriára külön-külön összeadjuk, és mintha a kategóriák helyi értéket jelentenének, leírjuk az így kapott számot. Hátulról előrefelé

30. Ez szerepel a dokumentációban például a szimbolikus linkről: „a chmod soha nem módosítja a szimbolikus linkek engedélyeit, a chmod rendszerhívás nem tudja módosítani az engedélyeket. Ez nem probléma, mivel a szimbolikus linkek engedélyei soha nincsenek használva”.
31. Onnan ismert, hogy valahogy a tudomásomra jutott, vagy éppen kitaláltam (de nem úgy, hogy „ls *” – ez utóbbihoz olvasási jog kellene).

szokták megadni, azaz a legutolsó számjegy az egyéb kategória jogait írja le, az utolsó előtti számjegy a csoportjogokat, hátulról a harmadik szám pedig a tulaj jogait[32]. Tehát az előzőek alapján a

```
chmod 751 fájl.bin
```

parancs az egyéb felhasználók számára 1 (azaz x), a csoporttagoknak 5 (azaz 4+1, tehát r és x), míg a tulaj számára 7 (4+2+1, azaz r, w és x) jogokat biztosít a fájl.bin-hez. Azaz mindenki futtathatja programként az adott fájlt, a tulaj és csoporttagok olvashatják is, míg a tulaj számára adott az extra módosíthatóság.[33]

Másik forma a jogosultságok beállítására az ún. szimbolikus forma. A felhasználói kategóriákat és a jogokat betűkkel jelöljük (pont azokkal, amiket korábban is használtunk):

Felhasználók:

– u = user, azaz a tulaj
– g = group, azaz csoporttagok
– o = other, azaz a „maradék”, mindenki más
– a = all, azaz mindenki, a fenti három együtt, egyszerűbben írva

Jogok:

– r = read, olvasás
– w = write, írási, módosítási jog
– x = execute/search, futtatási, keresési jog

Mivel a szimbolikus forma (a numerikussal ellentétben) erre is lehetőséget nyújt, ezért van egy harmadik tag. Milyen jellegű legyen a módosítás?

– = = jelentése, a bal oldalon álló felhasználói kategória pontosan a jobb oldalán álló jogokkal fog rendelkezni a sikeres művelet-végrehajtás után
– + = a bal oldali felhasználókategória a meglevő jogaihoz pluszban megkapja a jobb oldalon álló jogokat (az nem baj, ha ezek a jogok már eleve adottak)
– - = a bal oldali felhasználókategória meglevő jogaiból a jobb oldalon álló jogot elvesszük, az neki a sikeres végrehajtás után már nem fog a rendelkezésére állni (és ekkor sem gond, ha eredetileg sem volt neki)

Fentiekből legalább egyet, vagy vesszővel elválasztva többet is megadhatunk a chmod parancsnak a jogosultság állításához:

```
chmod u+w,go-x fájl.bin
```

Azaz a tulaj írási jogot fog kapni, míg mindenki mástól elvesszük a futtatási jogot. Vagy a numerikus formabeli példa:

```
chmod u+rwx,g=rx,o=x fájl.bin
```

(Annyit módosíthatnánk, hogy a tulajnál is egyenlőségjelet írunk, de szimbolikus formában a kívánt eredményt valószínűleg csak ennél bonyolultabban tudnánk beállítani.)

Természetesen nem kell mindig mindenki jogait állítani,

32. Van három speciális bit, amiket szintén a chmod-dal lehet állítani, és szintén az „ls -l”-ben látszanak. Ennél a numerikus megadásnál ezek szerepelhetnek a hátulról a 4. számjegyen.
33. Mindez csak részben igaz a futtathatóságra: héjprogramok (shell script) futtatásához az olvasási jogra is szükség van a futtatási jog mellett: nem véletlenül, mivel ezek olyan szövegfájlok, amelyek tartalmát a parancsértelmező fogja értelmezni, ha meg tudja őket nyitni olvasásra.

```
chmod u+w fájl.bin
```

csak a tulaj jogait bántja, a többihez hozzá sem nyúl.

chown

Mint elhangzott, a mindenkori tulaj és csoporttagok más jogokkal rendelkeznek. A tulajdonos megváltoztatható, ez ma Linux alatt általában rendszergazdai hatókör[34]. Erre szolgál a chown (change owner) parancs:

```
chown team5 fájl.bin
```

Innentől a team5 felhasználó lesz a tulajdonos (és neki lesz joga pl. a jogosultságokat átállítani). Mivel a fájl attól marad ugyanabban a könyvtárban, ahol eddig volt, szerencsétlen helyzetben ez ahhoz is vezethet, hogy a tulaj a tartalmazó könyvtár jogai miatt nem fér hozzá a tulajdonában levő fájlhoz.

chgrp

Ezzel a csoporttulajdonos állítható át: ezt a mindenkori tulaj és a rendszergazda változtathatja meg. (A tulaj csak olyan csoportok tulajdonába adhatja át a fájlját, amely csoportnak ő maga is tagja.)

```
chgrp staff fájl.bin
```

Immár a staff nevű csoport tagjaira vonatkoznak a csoportjogok.

umask

Az előző parancsok (chmod, chown, chgrp) már meglevő fájlok jellemzőinek átállítására alkalmasak. De milyen módon állítódnak be ezek a jellemzők a fájl létrehozásának pillanatában?

- tulajdonos: a fájl a létrehozó tulajdonába kerül
- csoport: a fájl a létrehozó éppen aktuális csoportjának tulajdonába kerül (az aktuális csoport az id parancs kimenetében a gid érték, illetve a groups parancs kimenetében látható első érték)[35]
- jogosultság: a fájl létrehozásakor a jogosultságot az ún. umask parancs segítségével lehet állítani

```
Az umask használata
```

Amikor egy alkalmazás létrehoz egy fájlt (típusa lényegtelen), egyúttal azt is megmondja az operációs rendszernek, hogy milyen jogokat szeretne adni ennek a fájlnak. A megadott jognál létrehozáskor nem lesz több, ez a maximum. (De utólag a fent tárgyalt chmod-dal módosítható.) Viszont ezt az alkalmazás által eldöntött maximumot csökkentjük az umask értékével. Az umask

34. Létezik rendszer, ahol a jelenlegi tulaj átadhatja másnak – persze mivel onnantól nem ő a tulaj, ezzel óvatosan kell bánni
35. Ha a könyvtáron be van állítva az ún. setgid-bit, akkor a csoportot a fájl a könyvtár csoportját örökli.

paraméter nélküli futtatáskor kiírja a jelenleg érvényben levőt, pl:

```
$ umask
0002
```

Jelentése a korábbi táblázat alapján: az alkalmazás által a fájl létrehozásakor javasolt jogokból mindenképpen el kell hagyni az „egyéb” felhasználói kategória írás jogát, azaz ha az alkalmazás pl. „rw-rw-rw-” jogokkal hozta volna létre a fájl, a végeredmény „rw-rw-r--” lesz.

Az umask-ot paraméterrel indítva, a továbbiakban létrehozandó fájlokra vonatkozó korlátot állítjuk be, pl:

```
umask 027
```

azt mondja, hogy a csoporttagoktól letiltjuk az írás jogot, egyéb felhasználóktól pedig minden jogot, azaz az előző példa alapján létrejövő fájl jogai „rw-r-----”-re változnának.

Speciális jogosultságkezelés

Attribútumok

Az EXT[234] fájlrendszerek rendelkeznek egy sajátos lehetőséggel, a fájloknak ún. attribútumokat lehet beállítani. Az attribútumok kezelésére szolgál az lsattr és a chattr parancsok. Előzővel lekérdezni, utóbbival beállítani lehet az attribútumokat. (A beállításhoz rendszergazda jogosultság szükséges.)

```
$ lsattr fájl
-------------e- fájl
```

A fenti kimenetben látható, hogy az „e” attribútum van beállítva (ez majdhogynem egyenes következménye a fájlrendszer belső működésének, így ezt kapcsolni sem lehet.)

```
$ chattr +a fájl1.txt     # append-only
$ chattr +i fájl2.txt     # immutable (semmilyen módon nem
  módosítható/törölhető)
```

A fenti parancsokban szereplő „a” attribútum jelentése: a fájl nem írható fölül, de hozzáírni lehet. Azaz pl. egy echo valami > fájl1.txt (átirányítás felülírással) parancs nem, míg az echo valami >> fájl1.txt (átirányítás hozzáírással) már működne. (Egyébként már törölni sem lehet a fájlt, vagy jogait változtatni.)

Az „i” attribútum pedig mindenféle a fájlra vonatkozó módosítást megakadályoz.

ACL-ek

Mivel a hagyományos tulaj, csoport, egyéb besorolása a felhasználóknak meglehetősen szűkös, ezért találták ki az access control list nevű találmányt. Gyakorlatilag egyes felhasználóknak és egyes csoportoknak megadható jogosultságról beszélünk.

A lehetőség Linux alatt fájlrendszerfüggetlen[36]. A parancsok: getfacl és setfacl. Mivel plusz felhasználók/csoportok kapnak jogot, és ezt a módosítás során ezt valahogy jelezni kell, ezért kicsit furcsa formában lehet megadni: az acl-ben jelzem, hogy felhasználó vagy csoport, utána hogy mi-

36. De kellhet hozzá az „acl” mount-opció

lyen nevű (felhasználó vagy csoport) végül pedig, hogy milyen jogokat állítok neki. Egy ACL beállítása és lekérdezése:

```
setfacl -m user:team5:rw fájl1.txt
getfacl fájl1.txt
```

A beállított ACL-ek törlése egyesével a -x opcióval, az összes ACL-t pedig a -b opcióval:

```
setfacl -x user:team5: lo
setfacl -b lo
```

Ha egy fájlon ACL bejegyzés létezik, az ls parancs -l opciója esetén a jogosultsági mezők végén egy pluszjel ezt jelzi:

```
ls -l
-rw-rwxr--+ 1 valaki team1 2 ápr   10 23:09 fájl1.txt
```

Kiterjesztett attribútumok

Minden fájlrendszer támogatja ún. kiterjesztett attribútumok (extended attributes, extattr, xattr) fájlokhoz hozzárendelését[37]. Ezek név:érték párok, amelyek különböző névterekben léteznek (azaz igazából mondhatnánk, hogy névtér.név:érték). Négyféle névtér létezik jelenleg, ebből a user szolgál a felhasználók által kezelhető adatok tárolására. Ezekbe az attribútumokba tetszőleges információ eltárolható. A kiterjesztett attribútumok kezelése a getfattr és setfattr parancsokkal történik. A következő példában a fájlhoz magához hozzáragasztjuk az MD5 hash-t. Így mindenféle adatbázis nélkül látható, ha valaki módosít valamit a fájlon (ha a módosítást végző nem ismeri ezt az apró trükköt):

```
setfattr -n user.md5 -v `md5sum fájl1.txt` fájl1.txt
```

Az attribútumok lekérdezhetők egyesével, ehhez ismerni kell a nevüket:

```
getfattr -n user.md5 fájl1.txt
```

vagy akár az összes attribútum:

```
getfattr -d fájl1.txt # dump all attributes
```

Természetesen törölhető is egy attribútum:

```
setfattr -x user.md5 fájl1.txt
```

Jelenleg nem ismerünk eszközt, amivel könnyedén észlelhetőek lennének a kiterjesztett attribútumok (mint amilyen pl. az ACL-ek esetén az ls parancs), viszont egy fontos jellemzőjükre felhívnánk a figyelmet. Természetesen a fájlok lokális, linuxos fájlrendszeren tárolása esetén használhatóak, de hálózati adatátvitelnél, vagy akár egy extattr-ot nem támogató fájlrendszerre másolásnál mindenféle figyelmeztetés nélkül elveszíthetjük őket. És mivel viszonylag könnyedén módosíthatóak, ezért a fenti példára biztonsági rendszert nem ajánlatos építeni.

37. Néhány fájlrendszer-típus esetén kellhet az user_xattr mount-opció

Folyamatok és kezelésük

A folyamat (process) egy elindított program még létező példánya. Egy PID (process identifier – folyamatazonosító) nevű, nem negatív egész számmal azonosítjuk. A 0-s azonosító fenntartott, az 1-es pedig egy speciális, az ún. init process ID-je. (Ez minden más felhasználói folyamat közös ősatyja. Vagy ősanyja, kinek hogy tetszik.) Egy rendszeren belül egy időben nem lehet két azonos PID-ű folyamat. A legelső (init) folyamat kivételével minden másik folyamat úgy keletkezik, hogy egy már létező folyamat létrehozza[38]. Az eredeti folyamatot a továbbiakban szülőfolyamatnak hívjuk (parent process, azonosítóját jellemzően PPID-ként emlegetik), a frissen létrejött neve: gyerek (child process). Természetesen a való élet szabályai itt is érvényesek, egy folyamat egyidejűleg lehet valamely másik folyamat gyereke, de mivel neki is lehet (akár több) gyereke, így egyben szülőfolyamat is. Azok a folyamatok, amelyek szülőfolyamatként a 2-s PPID-t mutatják[39], azok a rendszermag (kernel) részei (nem található önálló bináris ezzel a fájlnévvel), és csak adminisztrációs, ütemezési okokból látszanak folyamatként. Nevük: kernelszintű (vagy simán: kernel-), vagy még egyszerűbben: rendszerfolyamatok. A rendszerfolyamatokra nem érvényes semmilyen jogosultság kezelés. A folyamatok életük során használnak valamennyi CPU-erőforrást, memóriát is, terhelik a gép IO-alrendszerét. Egy folyamat megszűnésének van egy fázisa, amikor ún. zombifolyamatként létezik a rendszerben. Ekkor már tulajdonképpen nincs is; CPU-t, memóriát nem használ, egyedül a szülőfolyamat nem végzett el egy bizonyos adminisztratív feladatot. Ez teljesen normális állapot – de az már jellemzően programozói hiba, ha a zombi állapotú folyamat látványosan ottmarad.

ps

A folyamatokról információt vagy a /proc fájlrendszeren keresztül direktben, vagy pedig a ps paranccsal lehet szerezni. (Bizonyos ps implementációk szintén a /proc-ból nyerik ki az információkat, mások direktben a kerneltől.) A ps parancs egyik kellemetlen tulajdonsága, hogy rendkívül sok információt képes megjeleníteni meglehetősen változatosan szabályozható formában[40]. Ráadásul a a BSD-rendszerekkel és a zárt Unixokkal való kompatibilitási törekvések miatt legalább kétféle opciókezelése van. Ennek egyszerű következménye, hogy általában nem az egyes opciók jelentését szokták megtanulni, hanem helyette inkább „opciókupacokat”, és azok körülbelüli jelentését.

```
ps -ef
```

Az összes rendszerben futó folyamat (-e opció) „hasznosabb” (-f opció) jellemzőit megmutatja. Ezek között szerepel a folyamat „tulajdonosa” – akinek a jogosultságával fut az adott folyamat; látható a PID, az elhasznált processzoridő, a CPU-használatra vonatkozó adatok, és így tovább. Hasznos opciója a „-o”, aminek segítségével mi magunk határozhatjuk meg, hogy pontosan mely információkat akarjuk az egyes folyamatokról látni – ez nagyon hasznos parancsfájlok írásakor, hisz ebben az esetben sokkal könnyebb a minket érintő információkat összeszedni:

38. Erre jó a fork (illetve a vfork) rendszerhívás
39. Egyéb UNIX és UNIX-jellegű rendszert használók számára talán furcsa lehet, mert máshol a PPID=0 szokott lenni a rendszerfolyamat jellemzője
40. Ami egyszerre jelenti azt, hogy mely folyamatokat írjon ki, és azt is, hogy azok melyik adatát milyen sorrendben.

```
ps -e -o pid,user,cmd
```

A folyamatlistában látható, szögletes zárójelekkel bezárt folyamatok a fent említett rendszerfolyamatok. Ha pedig ezt látjuk a folyamat neveként, hogy <defunct>, az a korábban emlegetett, már erősen halálán van kategóriájú zombifolyamat. Sokat nem tudunk tenni ellene, meg kell próbálni a szülőfolyamatának elküldeni egy „Death of a Child”, azaz „Gyerekfolyamat megszűnt” jelentésű megszakítást (SIGCHLD).

kill, killall, pkill

Egy folyamat „kívülről” megszüntethető egy ún. megszakítás (signal) adott folyamatnak elküldésével. A megszakítások teljes listája lekérdezhető:

```
kill -l
```

Fontosabbak a HUP (1), INTR (2), QUIT (3), KILL (9) és TERM (15). A megszakításhoz szükséges a folyamat azonosítója, és hogy a megszakítást kezdeményező jogaival futó folyamatot akarjunk megszüntetni. Rendszergazda joggal természetesen mások folyamatai is megszakíthatók.

```
kill -2 1234
kill -n quit 1234
```

Elterjedt a név alapú forma, ehhez nem a folyamatazonosítót, hanem a futó program nevét kell megadni. Erre a pkill, vagy esetleg a killall parancsot érdemes használni. (A killall nagy hibája, hogy létezik olyan nem-Linux operációs rendszer, ahol a killall semmilyen paramétert nem vesz figyelembe, ellenben valóban minden folyamatot meggyilkol, ahogy azt a neve mondja is – így helyette talán érdemesebb a pkill-t megszokni.)

top

Míg a ps paranccsal egy adott pillanatbeli állapotot lehet lekérdezni, addig a top egy folyamatosan futó folyamatmonitorozó eszköz. Alapból a lista elején a legtöbb processzoridőt használó alkalmazások állnak, de ez már az indulásnál átváltható memóriahasználat alapú rendezésre.

```
top
top -m
```

Folyamatosan aktualizált listát kapunk. Futás közben interaktív parancsokkal a lista rendezési elve változtatható (N, M, P parancsok), de akár a folyamatok prioritása is változtatható (r), vagy akár meg is szüntethetünk egy folyamatot (k). (Újabban terjed a látványosabb htop nevű parancs használata a klasszikus top helyett.)

nice, renice

A folyamatok „jelentőségét” egy ún. „nice” érték határozza meg. Ezzel szabályozható a „CPU-éhsége”. Alacsonyabb érték magasabb prioritást jelent. Ez alkalmazás indulásakor öröklődik a szülőfolyamattól. Lehetőség van alacsonyabb nice-prioritással indítani egy kevésbé fontos alkalmazást:

```
nice -n 3 hosszan-futó-program
```

vagy akár futás közben is csökkenthető:

```
renice -n 3 -p PID
```

Rendszergazda joggal akár növelhető is a nice-prioritás (ehhez alacsonyabb érték tartozik), ezzel több processzor-erőforrás adható az alkalmazásnak.

Ha egy alkalmazás túl nagy CPU-igénnyel lép föl, szabályozhatjuk a folyamat CPU-használatát a cpulimit nevű paranccsal, ekkor a százalékos CPU-használatot írhatjuk elő:

```
cpulimit -b -l 25 -p 1234
```

A program a háttérbe visszahúzódva (-b opció) az 1234-es PID-ű folyamat CPU-igényét próbálja meg kordában tartani, és a processzor 25%-át (illetve egy ehhez közeli értéket) próbál tartani.

```
iotop, ionice
```

Ahogyan a futó folyamatok CPU és memóriahasználatát kényelmesen monitorozhatjuk a korábban emlegetett top (illetve a látványosabb htop) parancsokkal, ahhoz hasonlóan monitorozhatjuk az alkalmazások I/O használatát az iotop nevű paranccsal.

```
sudo iotop
```

A top-pal, htop-pal ellentétben az iotop jelenleg nem rendelkezik beépített súgóval, így egyszer érdemes elolvasni a dokumentációját, hogy kiderüljön, hogyan kell pl. az alapértelmezett rendezési elvet megváltoztatni[41].

Hasonlóan a CPU-használat szabályozásához, az I/O használat is szabályozható az ionice paranccsal. A folyamatok különböző IO-ütemező osztályban lehetnek, ezeken belül is különböző prioritással. A processzorhasználatot szabályozó nice értékhez hasonlóan az ionice értékre szintén igaz, hogy alacsonyabb érték jelenti a nagyobb prioritást.

```
ionice -c 2-n 0 sok-IO-t-igénylő-program
ionice -c 3 -n 7 -p kevés-IO-t-igénylő-folyamat-PID-je
```

Rendszergazdai jogokkal ún. valós idejű (real time) IO- illetve folyamatütemezés is beállítható.

41. Eléggé megdöbbentő módon a fejlécek között lehet sétálni a kurzor jobbra/balra gombokkal

A hardverek elérésének unixos, linuxos módja

Az operációs rendszer feladata a hardverelemek és a felhasználó közötti kapcsolat biztosítása. A UNIX rendszer – és így a Linux is – ezt a legtöbb eszközzel eszközfájlokon keresztül oldja meg. Ezzel a megoldással a különböző eszközök kezelése az általános fájlinterfészen keresztül történhet, úgy, hogy a programnak az adott eszköz sajátosságait nem kell ismernie. Így válik lehetővé például, hogy egy mágnesszalagok kezelésére írt programmal SSH-kapcsolaton át is lehet mentést készíteni, vagy egy lemezek másolására valóval hangfelvételt készíteni és visszajátszani.

Az eszközfájl egy speciális fájl, amelynek a műveleteit (mint az írás vagy az olvasás) a rendszermag nem a fájlrendszerrel valósítja meg, hanem más módon, tipikusan egy periféria működtetésével. Így érhetik el a felhasználói térben futó programok egy általános interfészen keresztül a lemezmeghajtókat, a terminált (billentyűzetet, egereket), a soros vonalakat, a hangkártyát, vagy éppen a kernelmemóriát.

Eszközfájlok típusai

Az eszközfájloknak két típusa van: a karakteres- és a blokkeszközök. A karakteres eszközökre az írás-olvasás byte-onként, pufferelés nélkül történik, és általában nincs lehetőség címzésre, ugrásra (ilyen például egy soros vonal). A blokkeszközöket pufferelve, blokkonként (eszköztől függően néhány kilobyte-os egységenként) kezeli a rendszer, és címezhetőek (ilyen például egy lemezmeghajtó).

Az karakteres- (c) és blokkeszközfájlok (b) ugyanúgy útvonal alapján érhetőek el, mint a többi fájltípus – a közönséges fájlok (-), a könyvtárak (d), a szimbolikus linkek (l), vagy éppen a csővezeték (pipe, p) vagy a socket (s). A két eszközfájltípust legegyszerűbben az `ls -l` paranccsal különböztethetjük meg – a kimenet első oszlopa a fájl típusát mutatja a felsorolás szerinti jelölésekkel:

```
$ ls -l /dev/sda /dev/ttyS0
  brw-rw---- 1 root disk    8,  0 dec    2 17:20 /dev/sda
  crw-rw---- 1 root dialout 4, 64 dec    2 17:20 /dev/ttyS0
```

Észrevehetjük a fenti példában is, hogy a listázáskor az ötödik mezőben, az eszközfájlokra nem értelmezhető fájlméret helyén két, vesszővel elválasztott szám szerepel. Ezek a „major”- és „minor”-számok, amelyek a kernel felé az elérni kívánt eszközt azonosítják.

Gyakran használt eszközfájlok

Pszeudoeszközök

A rendszermag számos olyan szolgáltatást is nyújt, amelyeket ugyan eszközfájlokon keresztül lehet elérni, azonban mögöttük nem egy hardvereszköz áll.

Gyakran felmerülő probléma az, hogy egy alkalmazás írni akar egy fájlba, azonban erre nincs szükségünk. Ilyenkor használjuk a nulleszközt, amely egy végtelen nyelő (sink). A szabványos helye a /dev/null. Az ebbe a fájlba írt adatot a rendszer eldobja. Olvasáskor egy nulla byte-os, üres fájlként viselkedik. Például ha egy HTTP lekérést kell csinálnunk, de az eredményére nincs szükségünk, a wget -O /dev/null http://drupalsite.tld/cron.php parancs a nulleszközbe menti a letöltött fájlt.

Hasonló eszköz a /dev/zero, amely íráskor hasonlóan működik a nullhoz, azonban olvasáskor nem üres fájlként, hanem végtelen forrásként viselkedik, mint egy végtelen hosszú, nullákból álló bináris fájl. Például a head -c 1024 </dev/zero >fájlom parancs beolvas 1024 darab nulla byte-ot a zero eszközből és a fájlom nevű fájlba írja.

A /dev/random és a /dev/urandom is végtelen forrásként működik, azonban nem nullákat, hanem véletlenszerű byteoket lehet belőlük olvasni. A head -c 1024 </dev/urandom >fájlom parancs így egy 1024 byte-os, véletlenszerű tartalmú fájlt hoz létre. A két eszköz között az a különbség, hogy az urandom csak pszeudovéletlen generátor, így sokkal gyorsabban olvasható, mint a valódi véletlen számokat biztosító random eszköz. Vegyük azonban figyelembe, hogy a rendszer entrópiájától függően sokáig tarthat akár kevés byte kiolvasása is a random eszközből. Ha például a /dev/sdx nevű lemezeszköz teljes tartalmát szeretnénk megsemmisíteni, akkor jó választás lehet a cat /dev/urandom >/dev/sdx parancs.

Statikus eszközfájlok

Az eszközfájlokat hagyományosan általános célú fájlrendszereken hozhatjuk létre, helyük a /dev/ könyvtár. Ezeket a fájlokat csak ritkán kell magunknak létrehoznunk, a telepítő elvégzi ezt a feladatot. Ha esetleg mégis hiányzik egy eszközfájl, és ismerjük a típusát, valamint major és minor számait (ezeket a rendszermag dokumentációjában, a devices.txt fájlban találjuk) akkor azt az mknod paranccsal hozhatjuk létre. Például az esetleg hiányzó /dev/null eszközt (mely egy karakteres eszköz (c), major száma 1 és minor száma 3) a root felhasználó az mknod -m 666 /dev/null c 1 3 paranccsal hozhatja létre (a 666 az eszközfájl védelmi kódja: mindenki írhatja és olvashatja). A hiányzó eszközfájlok létrehozását a statikus eszközfájlokat használó rendszereken a kívánt könyvtárban kiadott MAKEDEV parancs végzi el automatikusan.

A statikus eszközfájlok karbantartása több okból is nehézkes. A gépen rendszeresített összes eszközfájlnak léteznie kell akkor is, ha az eszköz éppen nincs csatlakoztatva – ellenkező esetben a root felhasználónak kell az eszközfájlokat létrehoznia és törölnie. A disztribúciók által szállított „mindent bele” moduláris kernelek esetében már az adott fájlnevekhez tartozó eszköz sem feltétlenül egyértelmű. Azonban a kis erőforrással rendelkező és ritkán változó hardverkonfigurációjú beágyazott rendszerekben, valamint más UNIX rendszerekben még előfordul, hogy a telepítéskor létrehozott, statikus eszközfájlokat használják.

Dinamikus eszközfájlok: udev

Az eszközfájlok létrehozásának és karbantartásának elkerülésére készült a devfs fájlrendszer, amely az aktuális disztribúciókban már ritkán fordul elő. Ez egy virtuális fájlrendszer, amelyben a kernel automatikusan hozza létre az eszközfájlokat. Számos hiányossága van: az eszközök állandó elnevezéseit, vagy a jogosultságok konfigurálásának lehetőségét sem biztosítja megfelelően.

A feladat megoldásának korszerű, általánosan elterjedt módja az udev használata. Ez egy olyan felhasználói térben futó szolgáltatás, amely a kerneltől értesítést kap az eszközök csatlakoztatásáról, és a konfigurációjában megadott szabályok alapján hozza létre, állítja be és törli az eszközfájlokat, valamint értesíti a felhasználói folyamatokat (például az asztali környezetet) erről.

Az udev által kezelt eszközfájlokat általában egy memóriában tárolt fájlrendszerre írjuk, mivel az operációs rendszer minden indítása után létrejönnek. A legkézenfekvőbb ilyen fájlrendszer a tmpfs, amelyet egyszerűen csatolhat a root felhasználó a `mount -t tmpfs none /csatolási/pont` paranccsal más célra is.

Mivel az elmúlt években sok disztribúció céljai között szerepelt a rendszerbetöltés gyorsítása, amit az udev szolgáltatás korai elindítása hátráltat, bekerült a Linux rendszermagba egy olyan, devtmpfs nevű öszvér-fájlrendszer, amely a devfs-hez hasonló megoldásaival automatikusan hozza létre az eszközfájlokat, de a jogosultságok beállításáról, a szimbolikus linkek létrehozásáról és az értesítésekről nem gondoskodik. Ezt a fájlrendszert az init folyamat csatolja, és csak később indul el az udev szolgáltatás. Egyes egyfelhasználós beágyazott rendszerek önmagában, udev daemon nélkül is használják ezt a fájlrendszert.

Reguláris minták és használatuk röviden

(A szabályos kifejezés angol neve: *regular expression*. Ezért hétköznapi használatban sűrűn használják a reguláris kifejezés, helyenként szabályos kifejezés elnevezést is, de valószínűleg még többször a „regex”, „regexp” neveket.)

A szabályos kifejezés egy olyan eszköz, aminek használatakor minta segítségével írhatók le különböző szövegek. Elsőre hasonlít pl. a fájlnevek megadásához használható helyettesítő karakterekre. De míg a dzsóker karaktereket elsősorban a parancsértelmezők használják, és gyakorlatilag csak fájlnevek kiválasztására alkalmazzák, addig a szabályos kifejezéseket elsősorban **nem** a parancsértelmezővel használjuk, hanem speciális alkalmazásokkal, és jellemzően fájlokon belüli szövegrészek kiválasztására.

Miért fontos megtanulni egy szövegkeresésre használható eszközt? Azért, mert Linux környezetben a legváratlanabb eszközökről derül ki, hogy képesek valamely funkciójukhoz regexet használni. Szövegszerkesztők, lapozóprogramok, különböző egyéb fájlmanipuláló alkalmazások – vagy éppen a web- vagy a levelezőszerver. Azaz akár felhasználó, akár rendszergazda valaki, biztosan belebotlik olyan feladatba, ahol jól jön ez a tudás.

BRE, ERE, PCRE

Szabályos kifejezéseket – azaz mintákat – általában úgy állítunk össze, hogy a mintában állhatnak önmagukat jelentő karakterek (elegánsan: literálok), és speciális (vagy meta-) karakterek: ez utóbbiak mást jelentenek, nem önmagukat (általában, de nem mindig több különböző karaktert; vagy karakterek speciális helyen való, vagy speciális darabszámú előfordulását). A UNIX/Linux-világban használt, regexképes eszközök sajnos abban nem egységesek, hogy melyek azok a karakterek, amik literálok, és melyek speciálisak. Vannak ugyan olyan – minden eszközre érvényes – szabályok, hogy például az angol ábécé minden betűje (26 kis, és 26 nagybetű), és a tíz számjegy (0-9) literálok, de az egyéb karakterek literális/speciális helyzete nem ennyire egyértelmű. Regex szempontból 4 csoportot lehet megkülönböztetni.

– BRE, Basic Regular Expression, azaz alapszintű regex. Az ebben definiált minták kezelését támogatja alapból pl. a grep és a sed nevű parancsok.
– ERE, Extended RE, azaz bővített készlet. A BRE-készlet bővítése, azaz speciális jelentést kap néhány olyan karakter, ami a BRE-készletben literál[42]. Ezt a „nyelvet” támogatja az awk; vagy a grep[43] és a sed[44] a -E opcióval indítva.
– PCRE, Perl Compatible RE, azaz a Perl nyelvben használt „nagyon kibővített” regex. Természetesen ezt használja maga a Perl (meg a Python, és a Ruby).
– Egyéb. Ez nyilván nem egy egyértelmű kategória, de nagyon sok olyan egyéb alkalmazás van, amely valamilyen regex „tájszólást” beszél. A többség a BRE valamilyen szintű kiterjesztését. (Pl. a vi szövegszerkesztő ismer olyan konstrukciót, amely a PCRE része; az expr nevű parancs

42. Az igazság az, hogy van, ami viszont éppen speciális a BRE készletben, de literál az ERE-ben. (Agyrém.)
43. A grep -E helyett sokszor használják az „egrep” parancsot. Ez utóbbi a régebbi forma.
44. A dokumentáció szerint a sed-nek nincs is -E opciója, hanem helyette a „-r” opció szolgál a bővített készlet használatára. Ennek ellenére működik. Érdeklődőknek: http://hup.hu/node/123278

majdnem BRE, de kicsit korlátozza azt, és í. t.)

Szabályos kifejezések használata

Használat szempontjából kissé kellemetlen, hogy a regexeket használó alkalmazásokat jellemzően a parancsértelmezőből indítjuk, és sok esetben magát a regexet parancssori paraméterként adjuk át az alkalmazásnak. Ez azért baj, mert van jó pár olyan karakter, amely egyszerre speciális karaktere a shellnek, és speciális jelentése van egy szabályos kifejezésben is. Ezért a félreértések elkerülése érdekébe rögtön az elején érdemes megjegyezni: ha parancssorban kell regexet használni, azt érdemes a shell elől mindig eltakarni. Például úgy, hogy aposztrófok közé zárjuk az egész mintát.

```
grep '[aAeEiI]*l.txt' /etc/passwd
```

Ha fenti példában elfelejtenénk kitenni az aposztrófokat, a parancsértelmező fájlnév-mintának tekintené az egyébként a grep-nek átadandó mintát, és ha pechünk van, sikerül is olyan fájlnevet találni (esetleg többet is), amelyre a mintát kicserélhetné. Fájlnévként ez a minta illeszkedik pl. az „Akol.txt”, „emel.txt” „Illatos virágszál.txt” fájlnevekre.

Illeszkedési szabályok

Illeszkedés vizsgálatánál pár szabály érvényes:

– az illeszkedés vizsgálatánál van amikor csak az illeszkedés ténye számít („megtaláltuk-e egyáltalán a keresett mintát bárhol?”), de van amikor fontos az is, hogy hol találtuk meg („csak a vizsgált szövegrész elején fogadható el a találat”), illetve esetenként fontos az illeszkedés hosszúsága. Csak literálokat tartalmazó minta esetén egyértelmű, hogy ha a minta 3 betűből áll, akkor illeszkedni vagy nem fog, vagy pontosan ugyanerre a 3 betűre, azaz 3 hosszúságban. Viszont metakaraktereket tartalmazó minta esetén az illeszkedés tetszőleges hosszú lehet, sőt az az extrém helyzet is előállhat, hogy a minta illeszkedik (azaz van találat), de az illeszkedés hosszúsága 0. (És ez nem egyezik meg azzal, hogy „nincs találat”.)

– egyetlen karaktert jelentő minta (ilyen minden literál, illetve néhány metakarakter) illeszkedik, ha az adott karakter megtalálható a vizsgált szövegrészben (pl. az „a” mint minta illeszkedik az „asztal”, „hajó”, „lámpa” szavakra, vagy a Kispál „Sika, kasza, léc” című lemezének címére, de nem illeszkedik a „Cseberből vederbe” kifejezésre).

– két vagy több literálból álló minta illeszkedik, ha mindegyik tagja illeszkedik, ugyanabban a sorrendben szerepelnek a mintában, mint a vizsgált szövegben, ráadásul az első literál illeszkedése után közvetlenül megtalálható a második literál illeszkedése, közvetlenül utána a harmadik, és í. t. (Azaz összetett minta esetén a rész-mintáknak illeszkedniük kell.) Példa: az „al” minta illeszkedik az „alma”, a „balra” szövegre, de nem illeszkedik az „ajtónál” vagy a „lampion” szavakra – első esetben külön-külön illeszkednek, de van a megtalált szövegek között más, a második esetben mind a kettő illeszkedik, közvetlenül egymás mellett is vannak, de nem jó sorrendben.

– ha egy minta többször is illeszkedne egy vizsgált szöveg esetén, akkor a legelső illeszkedést tekintjük megtaláltnak (van olyan eszköz és feladat amikor ez nagyon fontos szabály). Azaz a „fej” minta a „fejhús fejlövéssel” szöveg legelején (a fejhús szóban levőre) fog illeszkedni.

– ha egy minta illeszkedne rövidebb és hosszabb szövegre is (ezt ugye literálokkal nem tudjuk megtenni, de metakarakterekkel már igen), akkor a leghosszabb lehetséges szövegre fog illeszkedni – de az előző szabály erősebb, azaz „a legbaloldalibbak közül a leghosszabb” illeszkedés lesz érvényben (a metakarakterek bemutatása után erre is mutatunk példát)

Metakarakterek

Nézzük a speciális karaktereket! Figyelmeztetés: legtöbb regex megvalósítás esetén semelyik speciális karakter megadása nem teszi lehetővé az újsor karakterre való illeszkedést.

– Speciális jelentésű a . (pont) karakter. Ahol a mintában . áll, ott a megtalált szövegrészben tetszőleges karakter állhat. Példa: ha a minta a.b, akkor ez illeszkedik a l**amb**ada szóra (jelen esetben az „a” és „b” közötti „m”-et jelentette a „.”), vagy a k**alb**ász szóra (ahol pedig „l”-et jelentett a „.”), de nem illeszkedik a labda szóra, abban ugyanis az „a” és a „b” között nincs semmi, pedig a minta szerint kellene, hogy legyen.
– A \c két karakterrel leírt kombináció egyetlen c karakterre illeszkedik (azaz ezzel lehet egy metakarakter funkcióját eltakarni, literállá tenni). Példa: az a\.b már nem illeszkedik a lambadára, de a Léghajó.**a.b**álnák.fölött szövegre igen.
– A [...] azt jelenti, hogy a minta adott pozíciójában azok a karakterek állhatnak, amelyek a zárójelen belül állnak – és egy karakternek lennie kell. (Hasonlít a .-ra, csak annál szűkebb az értelme.) A szögletes zárójelen belül semmilyen karakternek nincs speciális jelentése (még a \ is elveszti speciális jelentését[45]), csak az alábbi kettőnek. Példa: az A[bcDEF] minta illeszkedik az **Ab**lak, vagy az **AD**ÉL szóra, de az alma már nem jó.
– Ha [^...] a minta, akkor a zárójelek között levő karakterekre pont nem illeszkedik, csak azokra, amik nem szerepelnek a felsorolásban. Ha a felsorolásban szerepelnie kell a ^-nak, akkor ez legyen bárhol, kivéve a nyitó zárójel után közvetlenül. tehát a [a^b] csak ezt a három karaktert jelenti, míg a [^ab] mindent, ami nem az „a” és nem a „b” (meg nem a soremelés). Példa: az A[^bcDEF] illeszkedik az **AS**ZTAL, **AB**LAK (mivel csak a kisbetű van kizárva) szavakra, de akár erre is „A mai nappal” (ebben az esetben az „A”-ra és az utána levő szóközre illeszkedett); de nem illeszkedik az ajtó szóra (mert nagybetűt írtunk elő).
– [a-z] és [^a-z] Azaz szögletes zárójelen belül írhatunk ún. karaktertartományokat, pl. a-m K-H, 5-8. Értelmezése: a kódjuk alapján egymást követő karakterek mindegyikét jelenti (a nyitó és záró tagot is beleértve). Az, hogy ezeket keressük, vagy épp ellenkezőleg pont ezeket nem keressük, az azon múlik, hogy szerepel-e a nyitó zárójel utáni első pozícióban a ^-karakter. Azaz: [0-9] – az összes számjegyet elfogadjuk találatként, [^A-Z] pedig az angol ABC összes nagybetűje kivételével minden mást elfogadunk találatként. Ha kötőjelet szeretnénk a felsorolásba betenni, akkor legbiztosabb módszer a záró zárójel elé közvetlenül, vagy a nyitó zárójel mögé – vagy közvetlenül, vagy ha negáltunk a ^ karakterrel, akkor e mögé írni. Azaz: [abc-] esetleg [^-M-S] Példa: az [a-zA-Z][0-9] minta mindent elfogad, ahol egy betűt egy számjegy követ. Pl. jó a „**B1**9-es bombázó” vagy a „Sinclair Z**X81** számítógép” szövegek, de már a „Commodore C-64” nem felel meg.
– Ha a minta első karaktere a ^ karakter (jelölése: ^RE), akkor az illeszkedést az RE rész határozza meg, de az illeszkedés helye csak a vizsgált szövegrész eleje lehet. Ez feladattól függő módon

45. Ez ([...\ ...]) konkrétan egy olyan szabály, amelyet nagyon sok programban sikerült hibásan implementálni. (A szabály leírását lásd: http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap09.html a 9.3.3 és a 9.3.5 pontok első bekezdését)

lehet, hogy a sor elejét jelenti (ha pl. soronként vizsgálunk valamit); lehet, hogy a mondat elejét jelenti (ha a feldolgozás mondat alapon történik); de jelentheti akár egy szó elejét is (ha a vizsgálat szó alapon zajlik). Legtöbb program sor alapú feldolgozást végez, tehát általában ez sor elejére „ragasztást" jelent[46]. Példa: az ^alma a mintát már csak akkor találja meg, ha egy sor az alma betűsorozattal kezdődik, még szóköz, tabulátor, egyéb nem-látható karakter sincs előtte.

– Ha a minta utolsó karaktere a $-jel, (jelölése: RE$), akkor a vizsgált minta csak a sor végén fogadható el. (hasonlóan a ^-karakterhez ez persze lehet sor, szó, mondat stb. – a feldolgozástól függő módon). Példa: a körte$ kizárólag akkor lesz találat, ha a sor végén a körte betűsorozat áll, se szóköz, se írásjel, se semmi egyéb nem állhat mögötte.
– Egy tetszőleges minta után álló *-karakter az adott minta tetszőleges darabszámú előfordulását jelenti – de csak a közvetlenül a csillag előtt álló minta-részre vonatkozik. A tetszőleges darabszám 0-t is jelenthet. Azaz: abc* illeszkedik az **abc**úg szóra (itt „c"-ből van 1 db,) de az **ab**sztinens és h**ab**patron szavakra is (amikor 0 db c-t talál), de nem illeszkedik a Hacsaturján, vagy a kacsatánc szavakra – mert hiányzik a kötelezően előírt „b".

Fenti lista korántsem teljes. Még a BRE készletben is vannak olyan metakarakterek, amelyek itt nem szerepelnek. Aki teljesebb leírásra vágyik, annak javasolt az egyik lábjegyzetben szereplő opengroups.org oldalon elérhető leírás – az maga a szabvány.[47]

Korábban szerepelt, hogy az illeszkedés mindig a legbaloldalibb találatot, de azok közül a leghosszabbat adja meg. Azaz ha a minta: aa*[48], és a vizsgált szöveg XaaYaaaZaaaa, akkor nyilván a leghosszabb találat a Z utáni 4 db a lenne. Viszont erősebb szabály, hogy a legbaloldalibbat kell megtalálni. Azaz az X utáni a. Vagy az X utáni aa. Mivel ezek közül a leghosszabb kell, ezért az X és Y közötti 2 db „a" lesz a találat. (Ezt amúgy pl. a Linuxokban meglevő GNU grep segítségével eléggé reménytelen ellenőrizni – sajnos eléggé félreérthető a kimenete, ezért tesztre kicsit jobban használható a később tárgyalandó sed, pl. a csere művelettel:

```
$ echo XaaYaaaZaaaa | sed -e 's/aa*/csere/'
XcsereYaaaZaaaa
```

Viszont ennél elsőre meglepőbb lesz az eredmény, ha a keresett mintát aa* -ról a* -ra változtatjuk:

```
$ echo XaaYaaaZaaaa | sed -e s/a*/csere/
csereXaaYaaaZaaaa
```

Mivel az illeszkedéshez a 0 db előfordulás is megfelel (ekkor illeszkedés van, csak épp 0 hosszúságban), ezért a sed-del végzett vizsgálat azt mutatja, hogy a legbaloldalibb előfordulásként a X előtti „semmit" – azaz a 0 db. „a" betűt találja meg és azt cseréli le. Utolsó példa:

```
$ echo aXaaYaaaZaaaa | sed -e s/a*/csere/
csereXaaYaaaZaaaa
```

Itt szintén a legbaloldalibb szöveget találja meg. Szintén jó lenne a 0 db „a", de itt már életbe lép a következő szabály, ami a leghosszabb megtalálását jelenti, így ugyanott van egy hosszabb találat is, az egy db. „a"-t tartalmazó – így ezt cseréli le.)

46. Sok regexet használó program esetén utal a dokumentáció arra, hogy „implicit anchoring" van érvényben. Ez azt jelenti, hogy minden mintát úgy értelmez az alkalmazás, mintha az elején lenne ez a ^-karakter.
47. Esetleg még ezen kívül Gábriel Ákos magyar nyelven elérhető (bár sajnos elég régi), a mai napig viszonylag kevés hibát tartalmazó tömör összefoglalója, pl. a http://www.szabilinux.hu/ismerteto/regexp.html oldalon. Ki hány hibát talál benne?
48. ERE esetén ezt a+ formában is írhatjuk, és azt jelenti, hogy néhány – de legalább egy db „a"

Szabályos kifejezést használó programok

Mint korábban szerepelt, sok program képes kezelni a szabályos kifejezéseket, ezek közül kettőről ejtünk itt szót. A grep és a sed használatának alapjait érdemes megtanulni (meg persze az AWK-t, a Perl-t, vagy akár kedvenc szövegszerkesztőnknek is utánanézhetünk, hogy keresés, vagy keresés-csere műveleteknél nem támogatja-e a regexeket.)

A sed (nagyon) alapjai

A sed egy nem-interaktív szövegszerkesztő, ami azt jelenti, hogy előre megmondjuk, hogy milyen feltételek teljesülésekor milyen módosításokat hajtson végre a paraméterként megkapott fájl tartalmával, a szerkesztés eredménye aztán a program normál kimenetén jelenik meg.

Egy sed parancs felépítése igen egyszerű:

– cím – azaz melyik sorral tegyük amit tennünk kell (sokszor elmarad, ez általában azt jelenti, hogy minden sorral végezzük el a műveletet. A cím lehet egy vagy két (vesszővel elválasztott) szám (az a sor, illetve az a tartomány), illetve /RE/ formában egy BRE szintaxissal megadott minta, amely azokat a sorokat címzi meg, amely sorokban a minta illeszkedik (itt is lehet tartományt megadni, akkor az első minta a kezdő sort, a második pedig a kezdő sor utáni sortól kezdve a tartományt lezáró sort adja meg). Például:

```
5
  13,24
  /^[A-Z]/
  /[0-9]$/,/kg/
```

– a cím mögött áll a művelet, amelyet a cím által kiválasztott sorokon végrehajtatunk.

– és ha a parancsnak szüksége van rá, akkor a paraméterek.

Noha nagyon sok szerkesztő parancsa van, ennek ellenére aki egyáltalán használja, az többségében 4 parancsra korlátozza a megtanulnivalót:

– valószínűleg leggyakrabban az s (azaz substitute, csere) parancsát használják

– ezt követi a d (delete, azaz törlés) parancs

– és a szöveg hozzáadására szolgáló a (append azaz mögéírás)

– és az i (insert, azaz elé beszúrás) parancsok

Pár példa kedvcsinálónak:

```
sed -i.bak -e ‘s/(C)/Copyright/g’ MyDocument.txt
```

A MyDocument.txt fájl minden sorában kicseréli a ‘(C)’ szöveget a Copyright szóra. Az amúgy nem szabványos, de Linux alatt használható -i.bak paraméter hatására az eredeti fájlt elmenti MyDocument.txt.bak néven, és a módosított szöveg visszakerül a fájlba. (Az i opció nélkül az eredmény az stdout-ra kerülne, nekem kellene átirányítással valamilyen másik fájlba tennem.)

```
sed -e ’/^[  ]*#/d’ -e ’/^[  ]*$/d’ /etc/ssh/sshd_config
```

Az SSH-szerver központi konfigurációs fájlját mutatja meg úgy, hogy kitörli az összes olyan sort, amiben csak megjegyzések vannak, illetve az üres (esetleg szóközöket, tabulátorokat tartalmazó) sorokat. Mind a két szerkesztő parancsban a szögletes zárójelek között egy szóköz és egy tabulátor karakter van – ez utóbbit bash-ban a CTRL-v TAB billentyűkombinációval lehet begé-

pelni (van olyan shell, amiben pedig \ TAB kombinációval).

```
sed -e '/^/a\
' /etc/profile
```

A képernyőre írja a /etc/profile nevű fájl tartalmát úgy, hogy minden olyan sor mögé, amire illeszkedik a minta[49], odaszúr egy db. üres sort.

```
sed -e ‘/^function /i\
\
\
‘ shell-functions.sh
```

Ez pedig a shell-functions.sh nevű hipotetikus fájlt jeleníti meg, de minden „function ” szöveggel kezdődő sor elé beszúr három üres sort. Így aztán a fájlban szereplő shell-függvények elkülönülnek kicsit egymástól.

A grep parancs

A grep arra jó, hogy szöveges fájlból gyorsan kikeressünk olyan sorokat, amely egy (vagy több) szabályos kifejezésre illeszkedik (vagy éppen nem illeszkedik). Gyakran használt opciói:

– -v ezzel írjuk elő azt, hogy a nem-illeszkedő sorokat szeretnénk megkapni

– -i kis-nagybetű érzékenység kikapcsolása (sokkal kényelmesebb, mint minden karakter helyett a [xX] formát írni pl.)

– -l a megtalált sorokat nem, csak a fájl nevét írja ki, amiben volt találat

(Természetesen nagyon sok egyéb opció van ezen kívül.) Fenti, a sed d funkcióját bemutató parancsot megcsinálhatjuk akár grep-pel is. Nyilván maga a két regex nem változik, csak kicsit a hívási mód:

```
grep -v -e ’^[  ]*#’ -e ’^[  ]*$’ /etc/ssh/sshd_config
```

Ha valakinek nem tetszenek a szabályos kifejezések, ne csüggedjen. Nem kell őket szeretni. Használni kell. Ha pedig valakinek tetszenek, akkor érdemes továbblépni, megismerni az itt kimaradt dolgokat, esetleg következő lépésként megtanulni előbb az AWK, utána a Perl lehetőségeit.

49. Mivel a minta a ^ („soreleje”) – márpedig minden sornak van eleje –, ezért ez gyakorlatilag minden sor mögé rak egy üres sort

A shell programozása minimalista megközelítésben

A parancsértelmező, mint programozási nyelv

A parancsértelmező egy speciális szintaxisú programozási nyelv értelmezője (interpreter) is, mely nyelv legalapvetőbb eszközeit a napi munka hatékony elsajátításához érdemes megtanulni. Vannak fanatikusai[50] és ellenzői – a többség egyszerűen nem veszi a fáradságot az alapok megtanulásához, ezért aztán a későbbiekben csak nyűglődik.

Mivel a shell interpreterként dolgozik, akár interaktívan, a parancssorba begépelve is lehet programocskákat írni és futtatni. Némi gyakorlás után ez egyre természetesebbé válik, és hamar eljuthatunk oda, hogy egy-egy feladat megoldásához 5-10-30 egyszerű parancsból álló „kódot" írunk – csak nem egy szövegszerkesztő ablakába, hanem a shell promptjánál. Nyilván az ily módon előállított kód nem nagyon szokott aztán bekerülni napi eszközeink valamilyen könyvtárban megmaradó tárházába, de sok esetben egyszerűbb ugyanazt (vagy majdnem ugyanazt) a 10 sort két héttel később újra begépelni, mint rájönni, hogy amikor gondosan elmentettük, vajon milyen névvel hivatkoztunk rá, hogyan is kellene paraméterezni, stb.

Az elején legfontosabb szabályként jegyezzük meg: míg interaktívan futtatva az egyes parancsokat láthatjuk az eredményt, a hiba-, vagy figyelmeztető üzeneteket, és ezek alapján dönthetünk a következő lépésről, addig ha parancsfájlt (angolul shell scriptet, vagy csak egyszerűen scriptet) írunk, akkor a parancsfájl futtatásakor látjuk ugyan az üzeneteket, de már nincs döntési lehetőségünk, a maximum, hogy ha figyelünk, esetleg megszakíthatjuk a program futását (esélyes, hogy jóval később, mint kellene). Ezért parancsfájlok írásakor legyünk körültekintőek, és lehetőség szerint készüljünk fel minden rossz bekövetkeztére.

Fenti szabály alkalmazása a gyakorlatban: mivel minden parancs a befejezésekor beállít egy ún. státuszkódot (hívhatjuk ERRORLEVEL-nek is akár), minden lehetséges alkalommal ellenőrizzük, hogy az előző lépés sikeresen, vagy sikertelenül fejeződött be. A státuszkód **0** értéke jelzi, hogy sikeres, és bármely 0-tól eltérő értéke a sikertelen befejeződést. Hasonlóan, az általunk megírt kódnál figyeljünk arra is oda, hogy szintén egyértelműen minden programocskánk jól állítsa be ezt a státuszkódot. (Erre szolgál a számparaméterrel ellátott exit parancs, pl: exit 2.)

Megjegyzések

A kettős kereszt (#, hashmark) karaktertől az adott sor végéig terjedő szöveget a shell megjegyzésnek tekinti. Nyilván interaktívan használva nem valószínű, hogy sok megjegyzést fűznénk a saját magunk által begépelt parancsokhoz, de parancsfájlok írásánál már erősen javasolt használni. Ráadásul van egy elég speciális funkciója. Parancsfájlok legelső sorába nagyon javasolt a következőt írni:

50. Érdemes elolvasni ezt a sokat sejtető című előadásanyagot: „A New Object-Oriented Programming Language: sh" http://www.usenix.org/publications/library/proceedings/bos94/full_papers/haemer.ps

```
#! /bin/sh
```

Fenti – amúgy megjegyzés – sor egy speciális jelzés az operációs rendszer számára, hogy ebben a fájlban olyan programkód található, melynek értelmezéséhez a /bin/sh nevű programot kell futtatni. Amennyiben programjaink kihasználják a bash speciális funkcióit, konstrukcióit, akkor természetesen helyette #!/bin/bash írandó.

A ~ (tilde) karakter

Egy tetszőleges szöveg elején álló ~ valamely könyvtárnév rövidített megadására szolgál. Ha önmagában áll, vagy közvetlenül mögötte egy /, akkor a felhasználó bejelentkezési könyvtárának nevét jeleni:

```
echo ~
```

eredménye

```
/home/team5
```

Ha pedig valamely egyéb szöveglánc következik, akkor a szöveg végéig, vagy az első /-ig terjedő részt felhasználói névnek tekinti a shell, és az adott nevű felhasználó bejelentkezési könyvtárának nevére cseréli le:

```
echo ~team5/.profile
```

eredménye

```
/home/team5/.profile
```

lesz.[51]

A „takaró” karakterek

Mint a legelső részben szerepelt, a begépelt parancssorban a szóközöknek, tabulátoroknak fontos szerepe van, az választja el egymástól az egyes részeket. Mondhatjuk, a szóköz és tabulátor a parancsértelmező számára speciális jelentéssel bír. Kezdők számára meglehetősen kellemetlen módon, ilyen „speciális” karakter nagyon sok van a shell eszköztárában. De mit lehet tenni, ha egy parancsban szeretnénk valahol ilyen speciális, jelentéssel bíró karaktereket használni? Pl. ki szeretnénk íratni egy szöveget, de a szövegben két szó között több szóközt használni. A probléma ugyanis, hogy az

```
echo alma       körte
```

parancs futtatásakor – hiába van az alma és körte szavak között több szóköz is, a parancsértelmező elnyeli a fölösleget, és úgy tesz, mintha pontosan egyet gépeltem volna be, azaz

```
alma körte
```

lesz a kimenet. Ezen probléma megoldására találták ki a shell ún. escape (vagy takaró) karaktereit.

51. Ennek az információnak az elfelejtése igen kellemetlen következményekkel járhat, lásd az alábbi fórumbejegyzést: http://hup.hu/cikkek/20120608/a_btrfs_elso_embere_elhagyja_az_oracle-t?comments_per_page=9999#comment-1671428

A legegyszerűbb a \ (azaz backslash: hanyattörtvonal, vagy rep-jel[52]). Ez a közvetlenül mögötte álló egyetlen karakter speciális jelentését szünteti meg (és egyébként a parancssorból, miután „megvédte” azt a fránya mögötte álló karaktert, gyakorlatilag nyom nélkül távozik). Azaz az előbbi parancsot ha jól akarjuk írni (azaz úgy, hogy megmaradjanak a szóközök), akkor ezeket szépen, egyesével takarni kell:

```
echo alma\ \ \ \ \ \ körte
```

Így a végeredmény tényleg úgy néz ki ahogy kellene:

```
alma       körte
```

Persze látható, hogy ez meglehetősen átláthatatlan parancsot eredményez, ezért csak ritkán, egy-két karakter esetén szokták használni. Helyette használható az egyszerű aposztróf, az ’ karakter. Ellentétben a rep-jellel, ebből kettő kell. Az első jelzi, hogy innentől nincs speciális jelentése a karaktereknek, a második pedig azt, hogy onnantól már megint igen. Azaz a szöveg elejének és végének jelzésére jó. Fenti példa tehát:

```
echo ’alma       körte’
```

formában is írható. Eredménye a már jól ismert:

```
alma       körte
```

A rep-jel és az aposztróf közös tulajdonsága, hogy minden speciális karakter elveszíti extra funkcióját.[53] Van egy harmadik, hasonló tulajdonsággal bíró karakter, ez az idézőjel, azaz a "-karakter. Működése nagyon hasonló az aposztrófféhoz, ugyanúgy párban kell szerepeljen, és a szöveg elejének és végének jelölésére szolgál. Azaz a fenti példánkat írhatjuk így is:

```
echo "alma       körte"
```

és ugyanúgy a megérdemelt

```
alma       körte
```

lesz az eredmény. Viszont idézőjelen belül nem minden speciális jelentésű karakter veszíti el a speciális jelentését, hanem csak a többség. Pontosan 3 karakter van, ami másként működik aposztrófok, és másként idézőjelek között, ez a három karakter a

$

`

\

Azaz a dollár-jel, a hanyatt-aposztróf – „magyarul”: backtick –, és a már jól ismert rep-jel. Ha ezen három karakter idézőjelpáron belül található, akkor ezek nem veszítik el speciális jelentésüket.[54] Látszólag persze egy meglehetősen jelentéktelen dologról beszélünk, de a későbbiek során látszani fog, hogy ezeknek a karaktereknek a „jó” használata meglehetősen sok helyen válik fontossá.

52. rep-jel, mert vissza-per
53. Lehet-e aposztrófok közé másik aposztrófot írni, illetve rep-jel mögé másik rep-jelet? Ha igen, miért nem? (Elnézést Karinthy Frigyestől a pofátlan lopásért.)
54. Megjegyezzük, hogy idézőjelen belüli rep-jel esetén a rep-jel **nem minden esetben** tartja meg speciális funkcióját, kizárólag teljesen logikus helyen: azaz dollárjel előtt, hanyatt-aposztróf előtt, rep-jel előtt és idézőjel előtt. (Ez tényleg logikus, de vajon Te is meg tudod magyarázni?)

Háttérben futtatás

Alapból egy általunk elindított parancs kisajátítja azt a terminált, ahol elindítottuk. Bizonyos feladatokhoz szükségtelen a folyamatos felügyelet, nyugodtan csinálhatna mindent a program automatikusan, mi meg valami mást. Ha a program fel van készítve a felügyelet nélküli futásra (nem kezd el futtában mindenféléket kérdezni a felhasználótól), akkor futtassuk háttérben. Ekkor az indítás után visszakapjuk a shell promptját, és indíthatunk újabb parancsokat (akár háttérben is). Használata:

parancs &

Átirányítások

Parancsok futtatásakor eddig nem nagyon foglalkoztunk azzal, hogy mi történik, elfogadtuk azt, hogy pl. egy echo parancs a képernyőre írja a paraméterként megadott szöveget. Ezt most pontosítsuk. A Linux parancsok[55] egy ún. normál bemenet, kimenet nevű „eszközön” kommunikálnak a felhasználóval. Más szóval, az echo nem a képernyőre, hanem egy ún. stdout (standard output; normál vagy szabványos kimenet; 1-es számmal is jelöljük) nevű helyre küldi az adatokat – ami viszont alapértelmezetten a képernyő. Ugyanígy, ha egy program adatot szeretne a felhasználótól elkérni, azt az stdin (standard input; normál vagy szabványos bemenet; 0-val jelöljük) nevű eszközön keresztül teszi, ami viszont alapbeállításban pont a felhasználó által használt billentyűzettől érkező adatokat jelenti. Ezen kívül létezik egy harmadik, neve stderr (standard error, normál vagy szabványos hibakimenet; 2-vel jelöljük). Ez utóbb a különböző hibaüzenetek megjelenítésére van fenntartva (sajnos ezt néhány programozó elfelejtette megtanulni és a hibaüzeneteket is az stdout-ra írják – mi ne tegyük ezt!).

Ha egyszer az stdout a képernyő, akkor minek ez a megkülönböztetés, hogy **nem** képernyőre, hanem stdout-ra írnak az egyes programok? Prózai oka van: ezek az alapbeállítások átállíthatóak, azaz meg lehet csinálni, hogy egy program – amelyik változatlanul az stdout-ra ír, a továbbiakban nem a képernyőre hanem egy fájlba, vagy akár egy másik programnak küldje át a kiírandó adatait. Ezt hívják összefoglaló néven átirányításnak. Talán leggyakrabban a kimenet átirányítása történik meg:

```
echo alma körte > fájl1.txt
```

Hatására a képernyőn semmi nem jelenik meg, viszont az alma körte szöveg bekerül a nagyobbjel mögött álló nevű fájlba. Ez a forma az ún. felülírásos (overwrite) üzemmód, azaz ha volt már korábban ugyanilyen nevű (azaz fájl1.txt) nevű fájl ebben a könyvtárban, akkor az eredeti tartalma elveszik.[56] Ha a fájl még nem létezett, akkor létrejön. Ráadásul a parancsértelmező először intézi el az átirányítás dolgait, és csak utána indítja el a programot: azaz ha

```
echo alma körte > fájl1.txt
```

volt a (félre)gépelt parancs, akkor a shell először eldobja a korábban esetleg már létező fájl1.txt tartalmát (vagy ha nem volt, létrehozza azt), majd utána amikor indítaná a parancsot, észreveszi, hogy ilyen parancs nincsen, ezt egy – az stderr -ra küldött hibaüzenettel honorálja, és a státuszkód beállítása[57] után befejezi az adott paranccsal kapcsolatos tevékenységet. (Azaz ha a parancs-

55. Precízebben: a karakteres felületről is futtatható programok többsége működik így, a grafikus alkalmazások már nem, de a karakteresek között is van egy-két kivétel.
56. Többet nem fogjuk külön kihangsúlyozni, de ehhez a jogosultsági rendszernek is lehet egy-két szava, azaz a továbbiakban végig azt feltételezzük, hogy az adott műveletek elvégzéséhez szükséges jogok rendelkezésre állnak.
57. Nem létező parancs futtatása 127-es hibakódot állít be.

sorban nem áll mögötte másik parancs – pl. ; -vel elválasztva –, akkor kiírja a promptot, ha van másik parancs, akkor kezdi annak a feldolgozását.)

A kimenet-átirányítás másik formája a >>, az un hozzáírásos (append) mód. Ekkor ha már létezik a fájl, akkor annak eredeti tartalma megmarad, és az eredeti adatok után azonnal tárolódik az átirányított adat. (Ha a fájl nincs, akkor ugyanúgy létrejön a fájl, mint a sima > átirányításnál.) Azaz feltételezve, hogy fájl1.txt az előző példa maradványa, fájl2.txt pedig eddig nem volt, a következő parancsok hatására

```
echo szilva banán >> fájl1.txt
echo dió mogyoró >> fájl2.txt
```

fájl1.txt két sort fog tartalmazni: az előző átirányítás alma körte, és a mostani szilva banán tartalmú sorait. Ezzel szemben fájl2.txt csak a dió mogyoró szöveget tartalmazza.

A kimenet átirányításhoz nagyon hasonló a hibacsatorna átirányítása, csak a >, illetve >>-jelek elé közvetlenül egy kettes számot kell írni, és így áll elő ez a forma:

```
cp file1.txt 2> fájl3.txt
cp . 2>> fájl3.txt
```

Mivel a cp legalább két paramétert vár (ráadásul külön opció nélkül az nem lehet könyvtárnév), mind a két parancs hibaüzenettel leáll. Viszont a hibacsatorna átirányítása miatt a hibaüzenet nem látszik a képernyőn, hanem eltárolódik a fájl3.txt nevű állományban.

A bemenet átirányítása sem sokban tér el, csak nagyobb-jel helyett kisebb-jelet kell használni, és a sikerességhez szükséges, hogy a hivatkozott fájl már létezzen:

```
cat < fájl3.txt
```

Ez a fenti két másoló parancs hibaüzenetét tartalmazó fájl tartalmát írja a képernyőre.

Gyakran előfordul, hogy egy parancs kimenetét egy másik parancs bemeneteként szeretnénk használni. Ekkor használhatjuk ezt a formát:

```
parancs1 > átmenetifájl ; parancs2 < átmenetifájl ; rm átmenetifájl
```

(A törlés a szemét eltakarítása miatt fontos, nyilván ha kell az adat, akkor hagyjuk ki.) E helyett egy speciális átirányítást javasolt használni, az ún. csövet, csőhálózatot (pipe, vagy pipeline). Azaz egyszerűen írjuk így:

```
parancs1 | parancs2
```

Ezzel a formával az első parancs kimenetét a második parancs bemenetével kapcsoljuk össze, ráadásul megspóroltuk az ideiglenes fájl létrehozásával és megszüntetésével kapcsolatos adminisztrációt.[58]

Elsősorban a kimeneti (stdout és stderr) átirányításokkal használatos még egy elég sajátos szintaxis. Az átirányítás után nem egy fájl neve áll, hanem a latin „és”-jel – azaz & és egy szám. Jellemzően ebben a két formában szokták használni:

```
echo Hiba >&2
parancs 2>&1 | less
```

Az első forma arra szolgál, hogy az echo (amely parancs alapból az stdout-ra ír) használható legyen hibaüzenetek megjelenítésére – ami viszont konvencionálisan az stderr. (Ez pont az, aminek fontosságára a fejezet elején felhívtuk a figyelmet.) Fenti trükkös átirányítással azt mondtuk, hogy a parancs szabályos kimenete menjen a szabályos hibacsatornára. A második parancs pont for-

58. Pl. nem kell aggódni, hogy véletlenül felülírunk valami fontosat, vagy épp ott hagyjuk a szemetünket.

dítva, a program szabályos hibacsatornáját csapja a szabályos kimenet mellé – ami viszont a csőbe megy, így mind a normál, mind a hibaüzenetek átkerülnek a lapozóprogramhoz, és nem kell félni attól, hogy esetleg az üzenetek kiszaladnak a képernyőről.

Végül: előfordulhat, hogy egy program futtatásakor mind a rendes, mind a hibaüzeneteket szeretnénk fájlban, méghozzá ugyanabban a fájlban eltárolni. Ezt két különböző módon is megtehetjük, mind a kettő meglehetősen nyakatekert:

```
program > fájl 2>&1
program 2> fájl >&2
```

Az eredmény ugyanaz, de csak ezek a megoldások adnak minden körülmények között használható eredményt.

Változók

A shell, mint a programozási nyelvek többsége, képes változók kezelésére. A változók neve betűk, számok és aláhúzás karakterből állhat, de nem kezdődhet számmal. Mint majdnem mindenhol Linux alatt, a kis- és nagybetűk itt is különböznek. Egy változóban tetszőleges érték tárolható, és egy shell-változónak nincs tipusa (illetve csak igen korlátozottan). Van pár olyan változó, amit maga a shell használ valamire, olyan is, aminek az értékét ő módosítja. Fontosabb változók pl. a SHELL, a PATH, a HOME, a LOGNAME vagy USER nevűek. A PATH kettősponttal elválasztva tartalmazza azokat a könyvtárneveket, amiket a shell végigkeres olyankor, amikor valaki elérési útvonal nélküli parancsot próbál indítani. Jellemzően valami ilyesmi az értéke:

```
PATH=/bin:/usr/bin:/usr/local/bin
```

Érdemes a többi környezeti változónak is utánaolvasni a parancsértelmező dokumentációjában. (És nem véletlenül szerepelnek nagybetűsen.)

Egy változó létrejön, amint értéket kap. Az értékadásnál ügyelni kell, hogy az egyenlőségjel egyik oldalán sem állhat szóköz. Azaz ezt írhatjuk:

```
a=5
```

(De ha ezt gépelnénk:

```
a= 5
```

akkor valószínűleg nehezen tudnánk megmagyarázni, hogy mi is történik pontosan.)

Ha valahol a változó értékére (azaz a tárolt adatra) vagyunk kíváncsiak, egyszerűen egy $-jel mögé kell írni a változó nevét, így:

```
$a
```

Nyilván ha tudni is szeretnénk, hogy mi ez, akkor használhatjuk pl. az echo parancsot:

```
echo $a
```

formában. A parancsértelmező az utasítás feldolgozása során fel fogja ismerni, hogy itt egy ún. változóhivatkozás szerepel, és még a parancs végrehajtása előtt ezt a hivatkozást kicseréli a változó adott pillanatban érvényes értékére. Azaz ha begépelem az **echo $a** parancsot, az először átalakul **echo 5** formára, és ez a parancs kerül végrehajtásra. Noha látszólag jól működik, ennek ellenére javasolt a változó-hivatkozásokat nem $v, hanem "$v" formában használni. Az alábbi pél-

da jól szemlélteti a problémát[59]:

```
$ v='alma    körte'
$ echo $v
alma körte
$ echo "$v"
alma    körte
```

Mint látható, csak akkor kapjuk meg a helyes eredményt, ha a változó-hivatkozás idézőjelek között áll. Ha az összes shell-változót szeretnénk látni, erre jó a set nevű parancs.

Korábban szerepelt, hogy minden parancs lefuttatása beállít egy ún. státuszkódot. Ez lekérdezhető egy speciális nevű, a shell által belsőleg kezelt „változó” segítségével. A változó neve: ? (ez karakter az általunk létrehozható változók nevében nincs megengedve), tehát ha egy parancs státuszára vagyunk kíváncsiak, akkor előbb futtassuk le az ominózus parancsot, majd így kérdezzük le a státuszát:

```
parancs
echo "$?"
```

Ha később szeretnénk felhasználni egy parancs státuszkódját, akkor helyette a parancs lefuttatása után tároljuk el a státuszt egy tetszőleges másik változóba:

```
parancs
errcode="$?"
```

Amíg felül nem írjuk, az errcode nevű változó ennek a parancsnak a visszaadott hibakódját fogja tartalmazni.

A shell-változók fontos tulajdonsága, hogy csak a változót létrehozó parancsértelmező konkrét példánya tud arról, hogy van ilyen változó (és mi az értéke). Egy speciális paranccsal ez egy csöppet módosítható. A parancs neve export, és segítségével elérhető, hogy az eredeti shell-példányon kívül az ebből a shellből később elindított egyéb programok ennek a változónak egy saját példányát megkaphassák (és akár ők is tovább örökíthessék). Beállítása egy vagy két lépésben:

```
a=1 ; export a
export b=2
```

szintaxissal történik. Fontos jellemzője az exportált (más néven: környezeti – environment) változóknak, hogy visszafelé nem lehet segítségükkel adatot átadni. Azaz ha az eredeti shellből exportáltam valamit, akkor ezt megkaphatja a shellből indított alkalmazás, de az alkalmazás már nem tud pl. egy módosított értéket visszajuttatni az őt elindító shellbe. Az összes környezeti változó listája névvel és értékkel lekérdezhető az env nevű paranccsal.

Parancssori paraméterek

Ha parancsfájlt hozunk létre egy feladat elvégzésére, valószínűleg szeretnénk ugyanúgy paramétereket átadni, mint ahogyan a rendszer gyári programjaival ez megtehető. Ehhez csak annyit kell tudnunk, hogy ha egy parancsfájl indításkor paramétereket kap, akkor ezek a paraméterek (a státuszkódhoz hasonlóan) speciális nevű „változók” értékére való hivatkozásként érhetők el. (Ezek nem változók a szó fent használt értelmében, de ugyanaz a szintaxis.) Azaz, egy parancsfájlon belül hivatkozhatok az első átadott paraméterre a $1, a másodikra a $2, a harmadikra a $3, stb. for-

59. Kivételesen jelezzük a promptot

mákat használva. Egyrészt itt is "$1" javasolható, másrészt tudni kell, hogy max "$9" – azaz a kilencedik paraméterre tudok ilyen könnyedén hivatkozni[60]. Ha elképzelhető, hogy 9-nél több paraméter van, akkor jó szolgálatot tehet a $# (ez az átadott paraméterek darabszáma – nyilván ha nem volt paraméter, akkor 0), másrészt hivatkozhatunk az összes átadott paraméterre az "$@" formával. (Tele van a net hibás dokumentumokkal és hibás programokkal amelyek e helyett a $* formát javasolják. Itt szólunk: a $* ugyanaz, mint a $@, de nem ugyanaz, mint akár a "$*" akár az "$*" – ráadásul csak az utolsó forma jelenti azt, amiről beszélünk: az összes átadott paraméter[61].)

Parancshelyettesítés

Amikor azt mondtuk, hogy a "$v" formát a shell speciálisan kezeli, helyette mondhattuk volna azt is: a parancsértelmező valamit helyettesített az általunk begépelt parancssorban. Ezt úgy hívjuk, hogy változóhelyettesítés. Más helyettesítés is létezik, legfontosabb az ún. parancshelyettesítés. Írhatjuk így: `parancs` (ez a régebbi forma)[62] és így is: $(parancs). Ha ilyet írunk valahova a parancssorba, akkor a parancsértelmező először lefuttatja a zárójelek (vagy hanyattaposztrófok) közötti parancsot, eltárolja a program kimenetét, majd az eredeti parancssorból kidobja ezt az egész konstrukciót, helyére rakja az eltárolt kimenetet, és az így létrejött parancsot is lefuttatja:[63]

```
$ tty
/dev/pts/3
$ ls -l /dev/pts/3
crw--w---- 1 zgabor tty 136, 3 dec    2 19:19 /dev/pts/3
```

Vagy egyszerűen bízzuk a shell-re:

```
$ ls -l $(tty)
crw--w---- 1 zgabor tty 136, 3 dec    2 19:19 /dev/pts/3
```

(Létezik még másmilyen helyettesítés. Vajon mi lehet ez?

```
$ echo $(( 2 * 3 ))
6
```

Neve is van: aritmetikai helyettesítés.)

Többirányú elágazás

Létezik egy parancs, amely arra szolgál, hogy egy feltétel igaz, vagy hamis voltát eldöntse. A parancs státuszkódjában 0 értéket visszaadva jelzi, ha a feltétel igaz, illetve nem-0 értékkel, ha a feltétel hamis. A parancs neve a sokat mondó test, használata szimplán:

```
test feltétel
```

A feltétel egyébként számok összehasonlítása, sztringek (szövegláncok) egyenlőség-vizsgálata, vagy fájlok jellemzőinek ellenőrzése. Ugyanez a parancs írható másik formában is:

60. Ha valaki szeretné, a könnyen áttekinthető "${10}" forma épp használható.
61. Aki nem hiszi, olvasgassa a man sh-t, vagy írjon kis programocskát, ami pl. ciklusban soronként kiírja ezen föntieket – aztán hívja meg a programját mondjuk az '1 2 3' "4 5" paraméterekkel.
62. Ez nem aposztróf, hanem az ún. hanyattaposztróf (backtick)
63. Itt is kiírjuk a promptot

```
[ feltétel ]
```

A parancsról, a különböző feltételek megadásának módjáról részletesebben a man test és a man sh dokumentációban lehet olvasni.

Shell-programok írásánál szükség lehet valamely feltételtől függő elágazásra. Erre az if és case nevű parancsok javasolhatóak. Az if egy parancs által visszaadott státuszkódtól függően hajt végre különböző dolgokat[64]:

```
if utasítás ; then
  ezt akkor hajtjuk végre
  ha 'utasítás' igaz (0)
  státuszt adott vissza
else
  ez kimaradhat, de ha van, a hamis
  státusz esetén hajtódik végre
fi
```

Ha kell, több if egymásba ágyazható, vagy akár az else helyett használható elif segítségével többirányú elágaztatás hozható létre.

Szintén többirányú elágaztatásra jó a case parancs, de annál egy szöveg és minták összehasonlítása határozza meg, hogy merre tovább. A minták megadásánál a másik fejezetben részletesen tárgyalt „joker-karakterek" használhatók.

```
case "$ANSWER" in
 [iI]* ) # a válasz i betűvel kezdődik
       csinálunk valamit
       ;;
 n* | N* ) # n-nel kezdődik
     másvalamit csinálunk ;;
*) # bármi egyéb esetén
    valami harmadikfélét csinálunk ;;
esac
```

Fenti részlet az ANSWER nevű változó tartalmától függően csinál 3 különböző dolgot. (Ha kihagynánk a legutolsó, a * -mintára illeszkedő részt, akkor kizárólag I-vel vagy N-nel kezdődő tartalom esetén történne bármi is.)

Ciklusszervező műveletek

Bash-ban 4-féle ciklusszervező művelet létezik: a for, while, until és az alig ismert (és szinte sosem használt) select parancsok. Ebből a for és while használata jellemző.

```
for i in alma körte szilva dió ; do
echo "$i"
done
```

Fenti programocska soronként kiírja az egyes gyümölcsök nevét. Működése: az "i" nevű ciklusváltozó sorban megkapja az in szócska és a ; közötti „értékeket (először az alma, aztán a körte, és í. t.), majd a do és a done közötti műveleteket hajtja végre. A példa remélhetőleg jól jelzi, hogy a for nem pont olyan, mint az ismertebb programozási nyelvekben. Nem kifejezetten a „számoljunk el

64. Azaz itt a „feltétel" a státuszkód nulla-vagy-nem-nulla volta (de az előzőleg tárgyalt test parancs segítségével klasszikus feltételvizsgálatos elágaztatást lehet csinálni)

1-től 100-ig" jellegű ciklusok megírására van kitalálva[65].

A while ciklus pedig (hasonlóan az if-hez) egy parancs státuszkódjától függően hajtja végre a ciklustörzsben szereplő műveleteket:

```
while read v ; do
   echo "$v"
done < /etc/passwd
```

Fenti parancs soronként olvassa az /etc/passwd fájlt, a beolvasott sor tartalmát eltárolja a v nevű shell-változóba, majd kiírja azt. Utána olvas tovább a fájlban. Mindezt addig teszi, míg a read nevű parancs hamis státuszt nem ad vissza, ekkor kilép a ciklusból. (Kitalálható, hogy a read akkor ad vissza hamis státuszt, mikor elér az olvasással a fájl végéig.)

(Nem-annyira) nyalánkságok

Utasítások csoportosíthatók a (p1 ; p2) vagy { p1; p2; } formában. (Pl. átirányításoknál érdekes lehet.) Egyszerűbb if konstrukciókat kiválthatunk a p1 && p2, illetve a p1 || p3 formával (az előbbi használható akkor, ha csak az igaz, a második akkor, ha csak a hamis ágban akarunk valami érdemlegeset csinálni).

Léteznek extrém egyszerű parancsok, mint pl. a true (semmit nem csinál, csak beállít egy igaz státuszt) vagy a false (no vajon?) Érdekesség, hogy a true parancsnak van egy rövidebb formája, a : (igen, a parancs neve: kettőspont).

Fentiek használatával kellően olvashatatlan kódot lehet előállítani, így használatuk erősen ellenjavallt – viszont mivel időnként azért szembejönnek, nem árt róluk tudni.

65. A `parancs` vagy $(parancs) formában is használható ún. parancshelyettesítés és elsősorban a nem túl hordozható seq nevű parancs segítségével egyébként lehet ilyen lépegetős for ciklusokat is írni Linux alatt, sőt ki lehet használni csak a bash-ban meglevő egyéb speciális konstrukciókat is. Ezeket a hordozhatóság érdekében nem javasoljuk.

Alapszintű hibakeresés

Hibátlan program nincs. Az axiómaként kezelt, az informatikával kapcsolatban rendszeresen elhangzó kijelentést lehet finomítani, de a lényegen nem változtat, ha e mellé mondjuk beleszúrjuk azt is, hogy „1-2 sornál/műveletnél/programkód-oldalnál hosszabb” esetről beszélünk. Sajnos a programok komplexitása nagyon megnőtt, a fejlesztési idő csökkent, és a tesztelési módszerek elméleti fejlődésével sajnos nem tartanak lépést a gyakorlatban elvégzett tesztek, így sok esetben marad a sommás, lemondó legyintés: hibátlan program nincs.

A probléma az, hogy ezeket a hibákat rendszergazdaként (sőt, sok esetben akár egyszerű felhasználóként[66] is) jó lenne elhárítani. Ehhez persze érteni kell, hogy mi történt, tudni kell hogyan működik a rendszer, ismerni kell a rendelkezésre álló eszközöket, képesnek kell lenni azokat használni, a dokumentáció (klasszikus man oldalak, info fájlok, beépített súgók, vagy akár az interneten elérhető *hogyan*ok[67]) olvasása sem hátrány. Csupa olyasmi, ami időigényes, sokak szemében elavult dolog, ráadásul sokakba belerögzült az a hibás kényszerképzet, hogy a számítógép az „csak úgy működik”, használatához nem kell tudás, ráadásul ha valami nem úgy működik, ahogy ő gondolja, azért a számítógép a hibás.

Az elejére általánosságban két alapszabály:

– olvassuk és próbáljuk meg értelmezni a hibaüzenetet

– nézzük meg a hibanaplókat.

Grafikus felület használata sok esetben nehezíti a hibakeresést, hiszen az emberek jelentős része már nem tudja, hogy pontosan mi is annak a programnak a neve, amelyiknek az ikonjára kattintva lesz neki hálózat. Meg böngésző. Meg elindul az az izé, amit nap, mint nap használ.

Ha tehát az a hiba, hogy hiába kattintunk a jól ismert ikonra, de nem történik semmi, vagy éppen nem az, ami szokott, még mindig segíthet maga a program. Lehet, hogy kiírja a hiba okát, csak nem látjuk. Ezért próbáljuk a következőt. Ha ismerjük a parancs nevét, máris rendelkezésünkre áll az egyik legősibb módszer: olvassuk el a hibaüzenetet. Nyissunk egy terminálemulátor ablakot, és gépeljük be a megfelelő parancs nevét, és lehet, hogy már készen is vagyunk.

Ha ez nem segít mert nem tudjuk a nevét, így elindítani sem tudjuk karakteres felületről, akkor sincs minden veszve. Ha már úgyis nyitva van a terminálablak, akkor adjuk ki a

```
tail -f ~/.xsession-errors
```

parancsot. Esetleg a megjelenő üzenetek láttán gyorsan nyomjunk pár ENTER-t, hogy legalább elkülönülhessenek a régebbi és az újabb üzenetek, majd miután ilyen jól felkészültünk, próbáljuk meg – immár a grafikus felületről – elindítani a problémás alkalmazást.

Primitívnek tűnik, de sok esetben már ezzel megtettünk mindent, ami a hiba megoldásához kellhet. Nyilván ezzel még nem hárítottuk el a hibát, de az üzenetekből talán máris rájövünk a megoldásra. Persze nem minden esetben elegendő elolvasni és megérteni a hibaüzenetet. Az se hátrány, ha a hibaüzenet értelmes, és valóban a hibáról szól. Sajnos ezen se a rendszergazda, se a felhasználó, se senki más nem tud érdemben változtatni, egyes-egyedül az adott kód fejlesztői.[68]

Hibakeresés során ne felejtkezzünk el a másik lehetőségről, amely eleve rendelkezésre áll. Tri-

66. Szerencsés élete van annak, aki az előforduló hiba esetén sikítva felállhat és abbahagyhatja a munkát, mondván „nem tudok dolgozni”.

67. Az angol *how-to* lassan terjedő magyar megfelelője.

vialitás, de sok program helyből naplózza működését. Nézzünk utána. Van-e saját logfájlja, és mi szerepel ebben a logfájlban? Netán a rendszer ez irányú szolgáltatását használja? Ekkor nézzük meg a rendszernaplókat. (Ez ügyben érdemes lehet utánanézni a rendszer syslog-konfigurációjának.)

Az alapok után nézzünk néhány egyszerűbb, egyéb technikát.

Ha adott a hibás program, kezdjük azzal, hogy kiderítjük: parancsfájlról, vagy éppen bináris programról van szó. Egyszerűen ellenőrizhető a

```
file /usr/bin/batch
```

paranccsal. Parancsfájloknál valami „POSIX shell script” vagy „Bourne-Again shell script” kezdetű eredményt kapunk, míg egy bináris alkalmazásnál már „ELF 64-bit LSB executable, x86-64, version 1” kezdetű lesz az eredmény. Ez a példa egy 64-bites operációs rendszerű, x86-architektúrájú gépen készült, értelemszerűen 32-bites rendszeren a szövegben is 32-bitnek kellene szerepelni, és az x86-64 helyett is állhat más.

Miért érdekes mindez? Egyszerűen más eszközök jók a parancsfájlok, és megint mások a binárisok hibás működésének felderítésében.

Héjprogram (shell script) esetén a legtriviálisabb eszköz a parancsfájl kézzel indítása. Két lehetőség is adott:

– sh -v parancsfájl paraméterek
– sh -x parancsfájl paraméterek

(Nyilván kombinálható is a kettő.) A „-v” opció hatására a shell minden egyes parancsot az adott utasítás végrehajtása előtt kiír a szabályos hibacsatornára – pontosan abban a formában, ahogyan a futtatandó állományban szerepel[69]. A „-v” és a „-x” opciók közötti különbség: az előbbi úgy mutatja a parancsokat, ahogyan szerepelnek a kódban, az utóbbi pedig már a különböző (változó, shell, aritmetikai, stb.) helyettesítések utáni formát. Komolyabb méretű programnál kicsit sok lesz ugyan valószínűleg a kimenet, de az már az átirányításoknál tárgyalt módon kezelhető:

```
sh -v parancs paraméterek 2>&1 | less
```

Van ennél egy jobb eszköz – ha a programhoz van írási jogunk, akkor módosítsuk a parancsfájlt oly módon, hogy a minket érdeklő részt keressük meg, és a gyanús hely elé rakjunk be egy

```
set -x
```

mögé pedig egy

```
set +x
```

parancsot. (Vagy -v/+v, ugyanúgy, mint a fenti sh paranccsal történő indításnál.) Ebben az esetben csak az ominózus részhez fogja a shell bekapcsolni a hibakeresési (debug vagy verbose) opciót. Ekkor persze már futtatható a megszokott

68. Példaként álljon itt egy ezt szemléltető történet. Még az internet korszak hajnalán, nagy élmény volt egy adatátviteli program kapcsán a futtatás során kapott „CRC error” üzenettel szembesülni. A lényeg az adatátvitelen van, tehát sejthető, mennyire jól esett, hogy jópár tesztfuttatás és az adatátviteli környezet erőteljes ellenőrzése után, még mindig „CRC error” volt az üzenet. Ekkor végül a rendelkezésre álló forráskód – amelynek léte persze nagyban segítette a hibakeresést – böngészése és módosítása után kiderült, hogy a valóságos hibához tartozó hibakód, hibaüzenet igazából az EPERM, azaz „Permission denied” – azaz a rendszerben volt egy hibás jogosultsági beállítás, ami megakadályozta a programot egy művelet végrehajtásában. Az ügyes programozó viszont nem érezte fontosnak a valós okot a felhasználó orrára kötni.
69. Ha valaki még emlékszik a @ECHO OFF sorra, annak nincs mit magyarázni

```
program paraméterek
```

formában.

Ha bináris program hibájának keresése a cél, akkor más eszközökkel érdemes nekiállni. Van, hogy már el sem indul az alkalmazás, pl. mert hiányzanak a futtatásához szükséges közös függvénykönyvtárak. (Elvben egy rendes, függőségeket is kezelni képes csomagalapú terjesztés esetén ilyen nem fordulhatna elő, de most nem az elméletről beszélünk.) Ekkor jöhet jól az ldd nevű parancs:

```
ldd /usr/bin/vi
```

Ekkor valami hasonlót kellene látnunk:

```
linux-vdso.so.1 =>  (0x00007fffa89ff000)
libtinfo.so.5 => /lib/x86_64-linux-gnu/libtinfo.so.5 (0x00007f3df1932000)
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f3df1713000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3df1352000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f3df114e000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3df1b78000)
```

Persze ha hiányzik valami, akkor helyette valami ilyen fog állni (már csak a hibás részt kiemelve):

```
libtinfo.so.5 => not found
```

A hibaüzenet önmagáért beszél, már csak meg kell keresni, hogy az adott fájl milyen csomag része, hol kellene lenni, és ki törölte le.

strace, ltrace

Hasznos eszköz lehet hibakereséshez az strace és ltrace nevű parancsok. Az strace segítségével majdnem a legalacsonyabb szinten, a rendszerhívások[70] szintjén követhető nyomon, hogy mi történik a program futása során. Ezzel szemben az ltrace kicsit magasabb szinten, a könyvtári függvények szintjén teszi ugyanezt. Indítási módjuk és pár fontosabb opciójuk is azonos:

```
strace program paraméterek
ltrace program paraméterek
```

Az opciók:

- -e kifejezés a nyomkövetés során miket szeretnénk figyelni. Pl. az strace-nél a -eopen opció az open nevű rendszerhívásokkal, míg a -otputs az ltrace-nél a tputs nevű függvényekkel kapcsolatos információkat naplózza. Természetesen vesszővel elválasztva több is megadható, de akár kizárni is lehet a listából elemeket.
- -p pid nem parancssorból indítjuk a programot, hanem már futó program nyomkövetését kérjük, a paraméter a követni kívánt folyamat azonosítója
- -o fájlnév a kimenetet nem a szabályos hibacsatornára, hanem a megadott nevű fájlba fogja írni a program
- -u felhasználónév rendszergazdaként indítva kérhetjük, hogy az elindítandó program más felhasználó nevében (és jogaival) fusson.

70. Rendszerhívásnak nevezzük az operációs rendszer magja (Linux alatt kernelnek hívjuk) által nyújtott, publikusan elérhető szolgáltatásokat,

A háttértárakkal (pontosabban a fájlrendszerekkel) kapcsolatos hibák közé tartozik a klasszikus probléma: betelik a fájlrendszer. Általában elég hamar feltűnik, az alkalmazások eléggé hangosan sírnak ilyen alkalmakkor. Ekkor az ember előszedi a háttértár foglaltságát megjelenítő df (disk free space) parancsot, és segítségével kikeresi, hogy melyik az a fájlrendszer, ahol hiba van (a 100%-os, vagy azt meghaladó(!) telítettség az hiba).

```
df -h
```

(Valahol a 90%-os telítettség környékén mindenképpen érdemes elgondolkodni újabb lemezterület rendszerbe építésén.) Miután megtaláltuk az ominózus fájlrendszert, jöhet a neheze: belépve a fájlrendszer csatolási könyvtárába, szisztematikusan kikeresni, hogy mely alkönyvtárak azok, amelyek a kelleténél több tárhelyet használnak. Erre alkalmas a du (disk usage) parancs, pl:

```
du -sh */
```

formában. Viszont az emberek szeretnek elfelejtkezni arról, hogy a fájlrendszerek adattároló területén kívül létezik egy speciális adatstruktúra, amely egyes fájlrendszereknél szintén korlátozott méretű: ez az ún. i-node tábla (az i-node vagy inode magyarul i-bög, de ebbe ne menjünk most bele[71]). (Minden fájl – akármekkora méretű, 1 darab i-node-ot elhasznál.) Az i-node tábla telítődése esetén a rendszer ugyanazt a „File system is full” hibaüzenetet adja, de a df magától nem jelzi a probléma okát. Erre szolgál a „-i” opció. Tehát keressük ki, hogy nincs-e véletlenül i-node-tábla telítődés:

```
df -i
```

Ha megtaláltuk, akkor már csak ki kellene törölni pár (tucat, száz, ezer) felesleges fájlt. A baj csak az, hogy hiába találtuk meg a bűnös fájlrendszert, nincs olyan kényelmesen használható parancs, ami kiírná, hogy melyik alkönyvtárban van nagyon sok fájl (mint amilyen az előző du volt, ami összeszámolta az elfoglalt helyet). A rendszergazda ilyenkor kénytelen előszedni a kismillió apró eszközt ami rendelkezésére áll, és megcsinálni magának.

Lépjünk be az ominózus könyvtárba, majd futtassuk le a következő borzalmat:

```
find . -xdev -type f | cut -d "/" -f 2 | sort | uniq -c | sort -n
```

Fenti parancs első fele (find) az aktuális könyvtár alatt kezd el keresni és minden közönséges fájlt kilistáz. A -xdev opció arra jó, hogy ne vizsgáljunk másik felcsatolt fájlrendszert, csak azt amelyikkel a probléma van. (A lista ./a ./b/c ./b/d jellegű sorokból áll.) Ezt a listát a cut paranccsal kicsit lecsupaszítjuk, minden sorból csak az első és (ha van) a második / közötti részt hagyjuk meg. Ezzel azt érjük el, hogy minden fájlnévből az marad meg, hogy a kiinduló könyvtár mely alkönyvtárában van (függetlenül attól, hogy közvetlenül abban, vagy annak valamely al-al-al-alkönyvtárában). Az első sort paranccsal ábécérendbe rendezzük az így kapott listát. Az ezt követő uniq megszámolja, hogy hány azonos, egymást követő sor volt (ezért kellett rendezni). És nem csak megszámolja, hanem ki is írja, minden sorban már csak egy számot, és egy könyvtárnevet. Végül az így kapott listát a második sort paranccsal immár nem ábécé-, hanem számsorrendbe rendezünk, és már meg is találtuk a nagyon sok fájlt tartalmazó könyvtárat. El lehet kezdeni takarítani.

71. Részletekért a Műszaki Kiadónál sokezer éve megjelent Kernighan–Pike: A UNIX operációs rendszer c. könyvét érdemes elolvasni (egyébként ettől függetlenül is érdemes)

E-közigazgatási Szabad
Szoftver Kompetencia Központ

MAGYARY PROGRAM

Nemzeti Fejlesztési Ügynökség
www.ujszechenyiterv.gov.hu
06 40 638 638

MAGYARORSZÁG MEGÚJUL

A projekt az Európai Unió támogatásával, az Európai Regionális Fejlesztési Alap társfinanszírozásával valósul meg.