BUDAPESTI MŰSZAKI EGYETEM
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék
(korábban: Folyamatszabályozási Tanszék)

Bevezetés a

UNIX

operációs rendszerbe

(oktatási segédlet)
4. bővített kiadás

összeállította:

Szeberényi Imre

Budapest 1998

Bevezetés a UNIX operációs rendszerbe

(oktatási segédlet)

Bevezetés

Ez a segédlet a Budapesti Műszaki Egyetem Irányítástechnika és Informatika Tanszék (korábban Folyamatszabályozási Tanszék) - a témában folytatott - több mint másfél évtizedes oktatói és kutatói munkájának ill. tapasztalatának felhasználásával készült.

A füzetben megjelenő dokumentációk eredeti UNIX dokumentációk rövidített fordításai. A fordítások egy részét a tanszéken készített "Bevezetés az SZM-4 számítógép UNIX operációsrendszerébe" című oktatási segédlethez Kőrösi István, Szabó Zoltán és Ketler István készítette 1982-ben. Akkor kicsit bújtatva jelent meg a UNIX oktatása a karon, hiszen a hallgatók "csupán" programozni tanultak az SZM-4-es számítógépen futó UNIX rendszer segítségével egészen 1989-ig.

Ezen segédlet jelenlegi formájában 1993-ban jelent meg először, amikor hosszabb szünet után ismét bekerült a Villamosmérnöki és Informatikai Kar oktatási programjába a UNIX, most már nem mint segédeszköz, hanem mint oktatási cél.

A rövid történeti áttekintés után szeretnék köszönetet mondani mindazoknak, akik a jegyzet összeállításánál hasznos tanácsokkal ill. értékes bírálatokkal segítettek. Külön köszönet illeti közvetlen kollégáimat Máray Tamást és Mohácsi Jánost a hálózattal foglalkozó fejezet elkészítésében kifejtett munkájukért.

Budapest, 1998. augusztus 21.

Szeberényi Imre

1. A UNIX rendszer fejlődése

Az első UNIX 1969-ben (tehát közel 30 éve) egy PDP-7 típusú gépre készült el a Bell Laboratóriumban. Készítői (Ken Thompson, Dennis M. Ritchie,) általános célú időosztásos rendszernek szánták. A fő cél az volt, hogy egy olyan rendszert készítsenek saját maguknak, amely kényelmes, és jól támogatja programfejlesztői munkájukat. Kezdetben nem piaci terméknek készítették, és bizonyára még álmodni sem mertek arról, hogy rendszerük később ekkora népszerűségre tesz szert. Az első rendszer még teljes egészében gépi (assembly) nyelven készült, de a sikereken és a kollégák elismerésén felbuzdulva 1973-ban az egész rendszert újraírták C-ben. (Csak a legfontosabb gépfüggő részek maradtak gépi nyelven.) A rendszer igen sok újdonságot tartalmazott, amelyeknek népszerűségét is köszönheti. Ezek közül a legfontosabbak:

- Általános időosztásos (Time Sharing) rendszer.
- Több felhasználós (Multi User) rendszer.
- Több taszkos (Multi Task) rendszer.
- Hierarchikus állományrendszere (file-rendszere) felszerelhető (mountable) kötetekkel bővíthető. Ez azt jelenti, hogy a különböző cserélhető és fix lemezegységeken levő állományrendszer az alaprendszerhez egyszerűen hozzákapcsolható.
- Egyetlen gyökér (root) van a rendszerben, a kötetek (adathordozók) számától függetlenül.
- A rendszer az állományokat sem tartalmuk sem nevük alapján nem különbözteti meg (azaz: nincs külön szöveg típusú állomány vagy bináris állomány, nem az állomány neve vagy kiterjesztése határozza meg az állomány funkcióját).
- A fa struktúrában keresztkapcsolatok (link) hozhatók létre.
- Hatékony differenciált állományvédelmi rendszer.
- Egységes beviteli/kiviteli rendszer az állományok, a fizikai berendezések és folyamatok között.
- Bevitel/kivitel átirányítás, csővezeték szervezés.
- Aszinkron folyamatok létesítésének lehetősége.
- Rugalmas, könnyen változtatható parancsnyelv.
- Hatékony, sokoldalú segédprogramok.
- Hardver független operációs rendszer.
- Nyílt rendszerarchitektúra.

1974-re készült el a UNIX első széles körben elterjedt verziója (V6), amely hamar népszerűvé vált. A V6 lényegében már mindazokat a fontos UNIX funkciókat tartalmazta, amelyeket a mai rendszereknél is megtalálunk. A rendszert kezdetben csupán csak oktatási intézmények kaphatták meg, ezek viszont ingyenesen és forráskódban. Ezért nagyon sok helyen módosították és továbbfejlesztették. Ez részben ***előnyös*** volt, mert

- sok jó, hasznos ötlet került bele,
- tényleg a felhasználók igényei szerint fejlődött, hiszen maguk a felhasználók fejlesztették.

Ugyanakkor ***hátrányos*** is volt, mert

- igen sok (több mint 10) eltérő verzió létezett (inkompatibilitás),
- a felhasználói programok gyakorlatilag hordozhatatlanná váltak.

Ma szerencsére nincs ilyen sok UNIX változat, mivel az igen sok irányzatból gyakorlatilag a 80-as évekre két fő ág maradt meg. Az egyik az AT&T fejlesztésű System X irányzat, a másik a Berkeley Egyetem által fejlesztett BSD X irány, amelyek egyre közelebb kerültek egymáshoz, és a gyártók egyre inkább a kompatibilitásra törekedtek. Így pl. a System V Release 4 rendszerbe több BSD kompatibilis rendszerhívás is belekerült. Az egységesítést segítette a két fő irányt alapul vevő POSIX néven ismert IEEE szabvány is, amelyet a gyártók többsége elfogad és betart. 1992-ben a Berkeley Egyetemen hivatalosan megszüntették a fejlesztést, bár a felhasználók a mai napig továbbfejlesztik ezt az irányt is (BSD386, NetBSD, FreeBSD). 1991 őszén a gyártófüggetlenség érdekében létrehozták az OSF (Open Software Foundation) nevű szervezetet, mely a gyártók együttműködésével kialakította a többnyire BSD alapú OSF/1 rendszert. Ma az OSF/1 képviseli a két hivatalos főirány egyik ágát, a másikat pedig a System V Release 4 (SVR4) rendszer.

Jelentőségük miatt meg kell említeni a szabadon terjeszthető rendszereket is, melyek fejlesztését lelkes felhasználók ill azok csoportjai végzik szerte a világon. Ezek közül a legjelentősebbek a már említett BSD fejlesztés folytatásaként megjelenő BSD/OS, NetBSD, FreeBSD rendszerek, és a POSIX irányvonalat követő LINUX rendszer. Ezek forráskódban szabadon hozzáférhető, és a PC-s felhasználók körében meglehetősen népszerű rendszerek, melyek mára kiforrott, stabil, megbízható verzióikkal felveszik a versenyt az üzleti UNIX változatokkal.

A következő táblázatban felsoroljuk az említett fő irányzatok legutolsó általunk ismert verzióját és az irányzatot képviselő rendszerek közül néhányat:

Irányzat	Legutolsó verzió	Néhány példa
AT&T System X rendszerek	System V Release 4.2	SCO ODT, AT&T System V, Sun Solaris, SGI IRIX
Berkeley BSD X rendszerek	BSD 4.4	Domain OS, ULTRIX, SunOS, BSD386, BSD/OS, NetBSD, FreeBSD
OSF rendszerek	OSF/1	DEC Digital UNIX, IBM AIX, HP-UX
LINUX rendszerek	LINUX 1.2	LINUX

1.1 ábra. A fő UNIX irányzatok

A UNIX "háború" végét talán mégis az jelentheti, hogy 1993 végén egy gyártó független szervezet, az X/Open kezébe kerültek a UNIX védjegyhez fűződő minősítési jogok.

A UNIX népszerűségének fontosabb okai:

- Nyílt rendszer. (Nem kötődik egyetlen konkrét gyártóhoz sem.)
- Mikroszámítógéptől mainframe-ig és szuperszámítógépig minden hardver platformra implementálható.
- Időosztásos működési elv.
- Felhasználóbarát, kényelmes környezet.
- Hatékonyság, nagy teljesítmény.
- Viszonylagos olcsóság.

2. A UNIX hierarchikus állományrendszere

A UNIX operációs rendszerben az alap-állományrendszer közvetlen elérésű mágneses adathordozón helyezkedik el. Az alap-állományrendszer tetszés szerint kiegészíthető ún. felszerelhető (mountable) kötetekkel. Ez azt jelenti, hogy más közvetlen elérésű adathordozókon lévő állományrendszer hozzákapcsolható az alap-állományrendszerhez.

A UNIX állományrendszerében az egyes állományok a következő típusúak lehetnek:

- közönséges állomány (plain file)
- speciális állomány (special file)
- katalógus (directory).

Ezek közül a katalógusok további - szintén e három típus közül való - állományokat tartalmazhatnak. (2.1. ábra, 'C' megjegyzés).

Az alap-állományrendszer kezdőpontja is egy katalógus, a "/" jellel jelzett gyökérkatalógus (root directory). A gyökérkatalógusnak kitüntetett szerepe van, mert minden további állomány ezen keresztül érhető el. Az alap-állományrendszer felszerelhető kötetekkel való kiegészítése nem más, mint egy adathordozó (esetleg több adathordozó, vagy egy adathordozó egy részének) összekapcsolása az alap állományrendszer egy állományával (amely csak katalógus, vagy közönséges állomány lehet). Ezen eljárás a "felszerelés" (mount). A UNIX-ban az alap állományrendszert tartalmazó adathordozót nem kell felszerelni, minden más adathordozót azonban igen.

2.1 Katalógusok

A katalógus az állományrendszerben egy olyan kitüntetett állomány, amely a benne bejegyzett nevek, és az azokhoz tartozó információk között adja meg a leképzést. Ezt az ún. i-node számmal éri el, mégpedig úgy, hogy minden bejegyzése egy nevet és a hozzá tartozó i-node számot tartalmazza. Az i-node szám egy fizikai helyet jelöl ki az adott adathordozón. Ezen a helyen egy jól definiált formátumú információcsomag található, amely az állomány összes jellemzőjét (fizikai elhelyezkedését, méretét, tulajdonost, védelmi kódokat, módosítási/létrehozási időket) megadja. Az i-node-okon a fizikai műveleteket maga a rendszer végzi, és az azokon való közvetlen műveletvégzést megtiltja a felhasználóknak. A katalógusok viszont egyszerű állományként elérhetők, azzal a megkötéssel, hogy még a super-user sem írhatja közvetlenül állományként azokat. A megfelelő

engedély birtokában azonban bárki olvashat egy katalógusállományt ugyanúgy, mint egy közönséges állományt.[1]

Egy állomány katalógusbeli bejegyzése tehát csupán az állomány nevét tartalmazza és egy mutatót az állományt ténylegesen leíró információra. Így egy állomány a katalógusbejegyzéstől függetlenül létezik, bár a gyakorlatban az utolsó rámutató hivatkozással együtt az állományt tartalmazó terület is felszabadul.

A gyökérkatalógus számos alkatalógust tartalmaz, amelyek mindegyikében újabb és újabb alkatalógus lehet. Így az állományokból egy hierarchikusan szervezett fa alakul ki.[2] Az egyes bejegyzések azonosítására az állománynév szolgál, amely teljes formájában a gyökérkatalógusból kiindulva az állomány megtalálásáig az összes alkatalóguson keresztül vezető utat írja le (útnév). Az állománynévre egyetlen megkötés az, hogy nem tartalmazhatja a "/" jelet, mert az az egyes alkatalógusok neveinek elválasztására szolgál. Más speciális karakterek (pl. *,",',!, stb) alkalmazása a parancsértelmező (**sh**) miatt célszerűtlen[3], de lehetséges[4]. A név hossza korlátos, a korai rendszereknél max. 14 karakter, de a mai rendszereknél ez lényegesen hosszabb[5] is lehet.[6] Hosszabb név megadása esetén a rendszer csak az első N karaktert veszi figyelembe, a többit figyelmeztetés nélkül elhagyja. Az állományrendszer nem, bizonyos programok azonban megkövetelik, hogy az állomány nevéhez kiterjesztést fűzzünk. Ennek formája: **név.kiterj**.[7] A kiterjesztések megadása az áttekinthetőséget növeli, így használata mindenképpen célszerű.

Az egyes katalógusok között a felhasználók szabadon mozoghatnak (természetesen a rendszer ellenőrzi a hozzáférési jogokat). Azt a katalógust, amelyben a felhasználó éppen tartózkodik, munka-katalógusnak (working directory) nevezzük. A felhasználónak a bejegyzések azonosításánál nem kell az állományok teljes útnevét kiírnia, ha a bejegyzés a munkakatalógusban található. A munkakatalógus váltását rendszerparancs biztosítja (**cd)**[8]. Célszerű a különböző részfeladatok elvégzésére külön-külön munkakatalógusokat használni, növelve az áttekinthetőséget és így elkerülni a névütközésekből származó problémákat.

Minden katalógusállomány legalább két bejegyzést tartalmaz, a **pont** és a **pont-pont** bejegyzést. A **"."** név magára a katalógusállományra vonatkozik. Így egy program a pillanatnyi katalógusállományt a **"."** alatt olvashatja anélkül, hogy ismerné az idevezető teljes elérési láncot. A **".."** név megállapodás szerint az

---

1 A POSIX előírásai szerint a katalógust nem kell elérhetővé tenni normál állományként.

2 Az MS-DOS állományrendszerhez való hasonlóság nem véletlen, mivel a DOS a UNIX-ból merítette az állományrendszer szerkezetének ötletét.

3 Nem látható kódú karakterek is lehetnek a névben, kivéve a 0 ASCII kódú karaktert.

4 Ajánlatos azonban a POSIX által ajánlott ún. portable neveket használni. Ezek csak betűket, számokat, aláhúzás- és kötőjelet tartalmazhatnak. Nem kezdődhetnek továbbá kötőjellel.

5 Pl. a BSD 4.3-ban 255 karakter lehet egy név maximális hossza.

6 A POSIX a név minimális hosszúságát írja elő, ez 14 karakter.

7 Az MS-DOS rendszerrel ellentétben ilyenkor a pont a név része, ami majd a különböző rövidítések használatánál (pl. *.*) fontos eltérést jelent.

8 Eltérés az MS-DOS munkakatalógus fogalmához képest, hogy a munkakatalógus a folyamathoz tartozik, és így minden folyamatnak saját munkakatalógusa van, amely új folyamat létrehozásánál öröklődik.

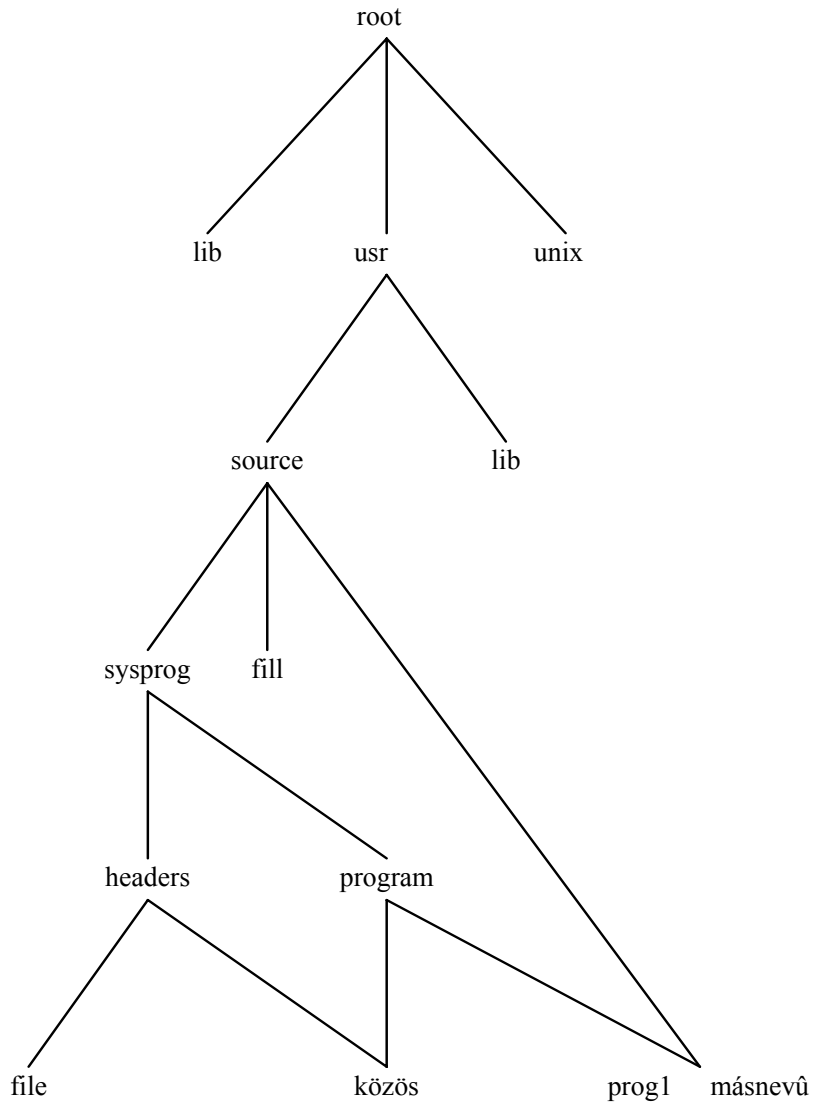
aktuális katalógus "szülő" katalógusára vonatkozik, tehát arra a katalógusállományra, amelyikben az aktuális katalógusállományt létrehozták. (A gyökérkatalógusnál a **"."** és a **".."** is a gyökérre mutat.) Szülőkatalógusnak azt a katalógust nevezzük, amelyben bejegyzésként előfordul az adott alkatalógus. Ennek szemléltetése a 2.1. ábrán található ('B' megjegyzés). A "." karakterrel kezdődő bejegyzések "láthatatlanok", azaz a katalógusról készített egyszerű listán nincsenek rajta.

A fa-struktúrából adódóan tetszés szerinti számú név azonos lehet, ha a hozzá vezető út különböző, hiszen az állománynév teljes formája ilyen esetekben jól megkülönböztethető. (2.1. ábra 'D' megjegyzés).

Mivel a katalógusok csak egy név → i-node összerendelést tartalmaznak, ezért nem kizárt, hogy egy katalógusból több néven, vagy esetleg több katalógusból azonos, vagy különböző névvel hivatkozzunk ugyanarra az i-node-ra. Ami viszont azt jelenti, hogy ugyanazt az állományt több néven is elérhetjük. Ezt a UNIX-ban link-nek hívják. Egyes rendszerek megengedik a különböző adathordozón elhelyezkedő katalógusbejegyzések közötti linket is, ez az ún. szimbolikus link[9] (symbolic link). Amíg az egyszerű vagy szoros linket (hard link) katalógusokra nem szokás (de lehetséges) használni, addig a szimbolikus linkeket igen gyakran alkalmazzák katalógusok között is. Linket két katalógusbejegyzés között az **ln** segédprogrammal lehetséges létrehozni. A katalógusok közötti link létrehozására csak a super-user-nek van joga, mivel ezzel a módszerrel hurkot lehet képezni a fában, ami viszont több program hibás működését eredményezheti. Természetesen nem szükségszerű, hogy a linkkel összekapcsolt bejegyzések neve ugyanaz legyen az összes katalógusban. Ezt szemlélteti az 2.1. ábra ('E' megjegyzés). Ha egy állományra több link is van, akkor az egyik bejegyzés által tartalmazott információt megváltoztatva az összes megváltozik. Az egy állomány különböző néven és helyen való bejegyzése számos előnyt eredményez:

- A programozói munkákban nélkülözhetetlen csoportmunka gyakorta igényel több helyre bejegyzett, de a valóságban csak egy helyen létező és a tulajdonos által karbantartott állományokat.
- A link szellemes lehetőséget biztosít jellegében hasonló, de paraméterezésében eltérő kiszolgáló feladatok implementálásában is. Erre jó példa **vi** és a **view**, ami egy program két névhivatkozással. Mivel a program induláskor megkapja saját nevét is, ezért ennek alapján másként viselkedhet. (Pl. a view nem enged írni az állományba.)

[9] A szimbolikus linkre a rendszeren belül egy kicsit eltérő mechanizmus van, ezért nem mindenben úgy viselkednek mint a normál linkek. Jó példa erre, a cd dir; cd .. parancssorozat eredménye, ahol a cd .. más helyre visz vissza attól függően, hogy a dir szimbolikus link-e. (Egyes parancsértelmezők ezt az ellentmondást feloldják, azzal, hogy a cd .. parancsot maguk értelmezik, de ez nem az operációs rendszer tulajdonsága.)



2.1. ábra. UNIX fa-struktúra

Megjegyzések:

A) a "prog1" állomány teljes neve: /usr/source/sysprog/program/prog1. A teljes névben az első "/" jel a gyökérkatalógus neve, a továbbiak pedig elválasztó karakterek, hogy a rendszer az egyes bejegyzéseket megkülönböztethesse.

B) Például a "program" katalógus (teljes nevén: /usr/source/sysprog/program) szülő katalógusa a "sysprog" (teljes nevén: /usr/source/sysprog). Az ábrán nem tüntettük fel a **"."**-tal kezdődő nevű bejegyzéseket, de ezek odaértendők az összes katalógusba.

C) Mint az ábrán látható, a "source" katalógus egyaránt tartalmaz közönséges állományt (fill) és katalógust (sysprog).

D) Az ábrán két "lib" bejegyzés is látható, azonban az egyik teljes neve "/lib", a másiké pedig "/usr/lib", így jól megkülönböztethetőek.

E) Az ábrán a "közös" nevű bejegyzésre két hivatkozás is van, az egyik a "headers" katalógusból, a másik a "program" katalógusból. Ezen kívül a "program" katalógus "prog1" bejegyzése által tartalmazott információra hivatkozik a "source" katalógus is, azonban más neve van az állománynak a "source" katalógusban ("másnevű").

2.2 Közönséges állományok

A közönséges állomány egy katalogizált adathalmaz. A rendszer minden állományt egyszerűen byte-ok véletlenszerűen címezhető szekvenciájának tekint. A rendszerfelület eltakarja az állományt tároló készülék fizikai tulajdonságait, pl. a diszksáv (track) méretét, blokkokat, cilindert stb. Az állomány mérete az általa tartalmazott byte-ok számával azonos. Az utolsó byte-ot az állományba történt írások során bekövetkezett "legmagasabb vízállás" határozza meg. A tárolási módszer érdekessége, hogy az állományok azon része, ahova még sohasem történt írás[10], nem foglal helyet az adathordozón. Az ilyen területről olvasáskor garantáltan nulla értékű byte-okat ad a rendszer. Az állomány számára előzetesen nem szükséges és nem is lehetséges helyet lefoglalni! Az adatfeldolgozásban szokásos rekord fogalom[11] teljesen hiányzik az operációs rendszerből és lényegében a szokványos programokból is. Egy szöveg típusú állomány például olyan ASCII kódú karaktersorozatként tárolódik, amelyben a sorokat a soremelés karakterek (LF - line feed) határolják. Ez a fajta tárolás nem csupán a helyigényt tekintve hatékony a fix-hosszúságú rekordokhoz képest vagy akár a karakter-darabszámmal jellemzett rekordokhoz képest, hanem éppen ez a legmegfelelőbb tárolási forma a legtöbb szövegfeldolgozó program számára is, amelyek túlnyomórészt karakterfolyamokkal dolgoznak. A leglényegesebb azonban az, hogy csupán **egyféle** szövegállomány ábrázolás van. A UNIX rendszer egyik előnyös tulajdonsága, hogy különböző programok igen nagy mértékben képesek együttműködni. Ez az együttműködés jelentősen megnehezülne, ha ugyanannak az információnak különféle ábrázolásai lennének. Ezen indokoknál fogva tehát a UNIX az állományokkal és különösen a szöveg típusú állományokkal kapcsolatban nem használja a rekordok hagyományos fogalmát. Vannak azonban olyan alkalmazások, amelyekben ez a fogalom hasznos. Egy program vagy egy önálló programcsomag jogosult bármilyen, általa hasznosnak tekintett adatábrázolási forma használatára. Az állományok struktúráját azonban az azokat használó programok maguk hozzák létre és ellenőrzik, nem pedig az operációs rendszer.

2.3 Speciális állományok

A UNIX-ban a fizikai berendezések is állományként vannak realizálva. Ez azt jelenti, hogy a fizikai berendezésekre állománynévvel lehet hivatkozni. A felhasználó számára teljesen azonos módon viselkednek az ilyen (ún. speciális) és a közönséges állományok. (Természetesen a berendezés fizikai jellemzőit figyelembe kell venni, például a sornyomtatóról nem lehet olvasni, stb). A speciális állományok szokás szerint a "/dev" katalógusban helyezkednek el, de bármely katalógusban lehetséges ilyen bejegyzést létrehozni. Minden, a rendszer által támogatott perifériához legalább egy ilyen állomány tartozik. Speciális állományok írása és olvasása ugyanúgy történik, mint a közönséges diszkes állományoké, de az olvasási, vagy írási kérések eredménye a megfelelő készülék aktivizálódása lesz. Speciális állomány tartozik minden egyes kommunikációs

[10] Lehet az állományban az író/olvasó pointert úgy mozgatni, hogy "lyukak" maradjanak.

[11] Hagyományos értelemben a "rekord" az információ egy olyan azonosítható egysége, amely vagy rögzített számú byte-ot, vagy egy darabszámot és azzal azonos számú byte-ot tartalmaz.

vonalhoz, diszkhez, mágnesszalagos egységhez és a fizikai főmemóriához is. Természetszerűleg az aktív diszkekhez és a memóriához tartozó speciális állományok védettek a nem megengedett hozzáférések ellen. A B/K készülékek ilyenfajta kezelésének háromféle előnye van:

- Az állomány és a készülék B/K annyira hasonló, amennyire csak lehetséges.
- Az állomány- és készüléknevek szintaxisa és jelentése azonos, tehát egy állománynevet váró programnak egy készüléknevet is átadhatunk paraméterként.
- A speciális állományokra ugyanaz a védelmi mechanizmus érvényes, mint a közönséges állományokra.

2.4 Állományrendszerek beiktatása és leválasztása

A UNIX állományrendszere a gyökérkészüléken elhelyezkedő gyökérkatalógusból indul ki, amelybe további, formátummal ellátott blokkorientált készülék iktatható be. Bár egy állományrendszer gyökere állandóan ugyanazon a készüléken tárolódik, nem feltétlenül szükséges az egész állományrendszer hierarchiának ezen a készüléken nyugodnia. Rendelkezésre áll a **mount** rendszerparancs amely, egy már létező állományt és egy másik készüléken elhelyezkedő kötetet[12] kapcsol össze. A mount parancs hatása az, hogy egy meglevő állományra való hivatkozások ezután az eltávolítható köteten lévő állományrendszer gyökérkatalógusára fognak vonatkozni. Gyakorlatilag a mount parancs a hierarchia fa egy pontját egy egész, új részfával cseréli fel (t.i. az elmozdítható köteten lévő hierarchiával). Természetesen a lecserélt pont (ami katalógus is lehet, és egyben egy részfa gyökere) a kapcsolat megszüntetéséig (**umount**) elérhetetlenné válik. A mount parancs végrehajtása után lényegében nem lesz különbség az eltávolítható köteten és a gyökér állományrendszerben található állományok között.

A különböző készülékeken található állományok azonos kezelése alól kivétel az állomány link. Különböző készülékek között csak szimbolikus link létesíthető. Erre a megkötésre azért van szükség, hogy elkerülhető legyen az a részletes könyvelés, amire egyébként szükség lenne annak biztosítására, hogy az állománykapcsolatok megszűnjenek az eltávolítható kötet lekapcsolásakor.

2.6 Pufferelés az állományok írásánál/olvasásánál

A felhasználó számára az állományok írása és olvasása egyaránt szinkronizáltnak és pufferezetlennek tűnik. Ez azt jelenti, hogy a read rendszerhívás után az adatok azonnal rendelkezésre állnak, write után pedig a felhasználó munkaterülete azonnal újra használható. A valóságban a rendszer egy meglehetősen hatékony pufferelő mechanizmust tartalmaz, amely nagymértékben lecsökkentheti egy állomány eléréséhez szükséges B/K műveletek számát. Tegyük fel, hogy egy olyan write rendszerhívás kerül kiadásra, amely egyetlen byte átvitelét írja elő. A rendszer először a saját puffereit tekinti át, hogy megállapítsa, az érintett lemezblokk nincs-e a tárban, ha nincs, akkor beolvassa a megfelelő készülékről. Ezután a szóban forgó

[12] Elvileg nem kizárt, de nem szokásos eset, hogy az így bekapcsolt adathordozón ne legyen állományrendszer. Ebben az esetben a bekapcsolt berendezés tartalma állományként jelenik meg a bekapcsolás helyén.

byte a pufferben lecserélődik, és egy bejegyzés készül a kiírandó blokkok listájában (ti. hogy a blokkot majd ki kell írni). Ezután megtörténhet a write-ból való visszatérés a felhasználói programhoz, bár a tényleges B/K csak később zajlik le. Másfelől ha egyetlen byte beolvasása van előírva, a rendszer először megvizsgálja, hogy a byte-ot tartalmazó lemezblokk nincs-e már valamelyik rendszerpufferben, ha igen, akkor a byte azonnal visszaadható. Ha nincs, akkor a blokk beolvasódik a pufferbe, és a byte onnan kerül a rendeltetési helyére. A rendszer felismeri, amikor egy program egy állomány szekvenciális blokkjaihoz nyúl hozzá és aszinkron módon előre olvassa a következő blokkot. Ez a legtöbb program futási idejét lényegesen lecsökkenti miközben, a rendszer terhelése ("overhead-je") csak kevéssé nő.

Azok a programok, amelyek a diszken levő adatblokk méretével[13] vagy annak egészszámú többszörösével egyező egységekben olvassák vagy írják az állományokat, előnyben vannak azokkal a programokkal szemben, amelyek ezt byte-onként teszik, de a nyereség nem túlzottan nagy. Ennek fő oka a rendszer-veszteség kiküszöbölése. Ha egy program ritkán fut, vagy nem végez nagymennyiségű B/K-t, észszerűen olvashat vagy írhat tetszőlegesen kis egységekben is.

2.7 Standard fa

A UNIX-ban egyes programok feltételezik, hogy bizonyos adatokat, állományokat egy megadott helyen találnak meg. Más esetekben viszont csupán a kialakult szokások miatt célszerű megtartani az ún. standard fa szerkezetét. A következőkben néhány, a helyes működés érdekében igen fontos katalógust ill. állományt ismertetünk:

/unix Magát a rendszert tartalmazó bináris állomány.

/bin Rendszerprogramok elhelyezésére szolgáló katalógus.

/usr/bin Közérdekű felhasználói programok elhelyezésére szolgáló katalógus.

/usr/adm Olyan állományok találhatók benne, amelyek a UNIX elszámolási rendszerével kapcsolatosak. Pl. a "wtmp" állomány, ami minden felhasználó be- és kijelentkezésének időpontját tartalmazza.

/etc A rendszer működéséhez, illetve karbantartásához szükséges állományok, programok találhatók itt.

/etc/passwd
Ezen állományban a felhasználók azonosításához szükséges adatok találhatók, a rendszerbe csak azok léphetnek be, akik szerepelnek ezen állományban.

/tmp Ideiglenes állományok elhelyezésére nyílik itt lehetőség;

/dev A speciális állományok katalógusa.

[13] Ez rendszerint 512 vagy 1024 byte.

Néhány jellegzetes UNIX periféria név:

dev/tty* A terminálok nevei. a "*" helyén bármely karaktersorozat állhat a gép kiépítettségétől függően.

/dev/hd* A "*" helyén bármely karaktersorozat állhat. A megadott szám a nem cserélhető mágneslemez egység (diszk) neve.

/dev/fd* A "*" helyén bármely karaktersorozat állhat. A megadott szám a cserélhető mágneslemez egység (floppydiszk) neve.

/dev/lp* A sornyomtató neve.

/dev/mem Az állományként elérhető memória neve.

/dev/null Különleges rendeltetésű állomány neve. Rendeltetése kétféle lehet: ha olvasunk belőle, akkor azonnal az állomány végét érzékeljük, ha pedig írunk bele, akkor korlátlanul elnyel mindent anélkül, hogy bárhol is tárolná.

2.8 Dokumentáció szerkezete

Hagyományosan a UNIX leírások 8 fejezetre (kötetre) oszlanak, amelyeket 1-8-ig sorszámokkal láttak el. A könnyebb tájékozódás érdekében egyes rendszerekben a főfejezeteken belül néhány speciális alfejezetet is megkülönböztetnek, amelyeket egy betűvel jelölnek.

Az egyes fejezetek a következő információkat tartalmazzák:

1. Az általánosan használható programok leírását.
   - 1 Standard parancsok.
   - 1C A kommunikációs parancsok.
   - 1G Grafikus programok.
2. A rendszerhívások leírását.
3. A felhasználói szintű könyvtári függvények leírását.
   - 3 Standard C funkciók.
   - 3N Hálózati funkciók.
   - 3R Távoli eljáráshívások (RPC) funkciói.
   - 3S Standard B/K funkciók
   - 3C Más rendszerekkel való kompatibilitást megvalósító funkciók.
   - 3M Matematikai könyvtár funkciói.
4. Speciális állományok, azaz a fizikai eszközök leírását.
   - 4 Lokális eszközök.
   - 4M Stream modulok.
   - 4N Hálózati felületek.
   - 4P Protokoll családok.
   - 4F Hálózattal kapcsolatos eszközök.
5. Állományformátumok leírását.
6. Játékok leírását.
7. Különféle leírásokat. Pl. a dokumentációk nroff makróinak leírását.
8. A rendszer karbantartásához szükséges információkat.

Fontos tudni, hogy több fejezetben is találhatók azonos nevű leírások, amelyek esetenként még témájukban sem kapcsolódnak egymáshoz. Pl.

write(1) - Segítségével lehet üzenni egy felhasználónak.
write(2) - A write rendszerhívás.

A kézikönyveket on-line módon is lekérhetjük, kereshetünk bennük a man parancs segítségével. A man parancs fontos szolgáltatása a kulcsszó szerinti keresés (ld. M. függelék).

2.9 Kapcsolódó parancsok

Röviden felsoroljuk azokat parancsokat, amelyek kapcsolatosak az állományrendszerrel. A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai. (A felsorolt parancsok egy részének felhasználói leírása az M. függelékben megtalálható).

cat(1)	-	állományok összemásolása
cd(1)	-	katalógusváltás
cp(1)	-	állomány másolása
dd(1)	-	állomány másoló és konvertáló program
df(1)	-	lemezek foglaltsági állapota
du(1)	-	diszkfoglalás kiíratása
find(1)	-	állomány keresése a fában
file(1)	-	állomány típusának meghatározása
fsck(8)	-	állományrendszer ellenőrzése
ln(1)	-	új állományhivatkozás készítése
ls(1)	-	katalógus listázása
mount(8)	-	kötetek bekapcsolása az állományrendszerbe
mkdir(1)	-	katalógus létrehozása
mv(1)	-	állomány átnevezése/áthelyezése
pwd(1)	-	munkakatalógus nevének kiíratása
rm(1)	-	állomány törlése
rmdir(1)	-	katalógu störlése
umount(8)	-	kötetek lekapcsolása az állományrendszerről

3. UNIX védelmi rendszere

Mint minden többfelhasználós, multiprogramozott rendszer, a UNIX is védi az egyes felhasználók **állományait** és éppen futó **folyamatait** a többi felhasználótól.

3.1 Felhasználók azonosítása

Minden egyes felhasználó egy felhasználó azonosító számmal (továbbiakban **uid** = user identifier), és egy vagy több csoport azonosító számmal (továbbiakban **gid** = group identifier) rendelkezik. A felhasználók azonosítása kizárólag az uid alapján történik. A gid-et a csoportmunka megkönnyítésére definiálja a rendszer. A gid segítségével a felhasználók egy részének több jogot lehet biztosítani, mint a többieknek. A felhasználók uid-je, és az alap gid-je a /etc/passwd állományban van rögzítve. Egy felhasználó a saját csoportján kívül más csoportokba is tartozhat. Egyes UNIX változatokban akár egyidőben is. A csoportok és a felhasználók összerendelését a /etc/group állomány tartalmazza. Az uid és a gid alapján egy adott bejegyzés szempontjából háromféle felhasználó létezik:

- Azon felhasználó, akinek az uid-je megegyezik a bejegyzés uid-jével, az a **tulajdonos** (owner).
- Akinek az uid-je más, de valamelyik gid-je megegyezik a bejegyzés gid-jével, az a **csoporttárs** (group).
- Akiknek pedig az uid-je és a gid-je egyaránt eltér a bejegyzés uid- illetve gid-jétől, azok a **többiek** (others).

Létezik még egy felhasználó, az úgynevezett kitüntetett felhasználó (továbbiakban **su** = super-user). A su-ra nem vonatkoznak a felhasználók által beállított hozzáférési jogok[14], a rendszer még a tulajdonosnál is bővebb lehetőségeket engedélyez számára[15]. A su a rendszergazda. A rendszer annak alapján ismeri fel, hogy az uid-je 0. Jogai gyakorlatilag korlátlanok, éppen ezért igen körültekintően kell használni ezt az uid-et, mert egyetlen rossz mozdulattal az egész rendszert tönkre lehet tenni.

3.2 Védelmi kódok

A létrehozott bejegyzések hozzáférési jogkörét a tulajdonosok egyedileg állíthatják be. Egy adott bejegyzésre háromféle engedély adható: olvasási, írási és végrehajtási (katalógus esetén ez utóbbi helyett keresési). Ezek mindegyike külön-külön állítható be magára a tulajdonosra, a csoporttársra és a többiekre. A hozzáférési jogok legegyszerűbben a **chmod** paranccsal módosíthatók (ld. függelék).

Az **olvasási** jogkör szabályozza, hogy van-e joga az adott felhasználónak az állomány tartalmának megtekintésére (illetve a katalógus tartalmának kilistázására). Az olvasási jog önmagában még nem jogosítja fel a felhasználót arra, hogy a bejegyzés tartalmát megváltoztassa, ehhez az írási jog szükséges.

Ha egy felhasználó rendelkezik **írási** joggal egy bejegyzésre, akkor bővítheti, kicserélheti, megváltoztathatja az állomány tartalmát (illetve létrehozhat, törölhet bejegyzést katalógus esetén).

A **végrehajtási** jog (amely csak állományra vonatkozhat, ha katalógusra vonatkozik, akkor keresési jognak nevezzük) azt jelzi, hogy az adott felhasználó futtathatja-e az állományt. Maga a végrehajthatósági jog önmagában még nem jelenti azt, hogy az állomány ténylegesen futtatható kódot tartalmaz. A parancsértelmező program ehhez megnézi az állomány első szavát. Ha ez a szó egy mágikus szám (magic number), akkor az állomány valóban futtatható kódot tartalmaz (ezeket a számokat a fordítóprogram automatikusan helyezi el - egyéb információk mellett - az állományban). Ha nem tartalmazza ezen számok egyikét sem, akkor parancs-állománynak tekinti, és megpróbálja végrehajtani az abban leírt parancsokat. A végrehajthatósági jog tehát csupán azt jelenti, hogy van-e jogunk futtatni az állományban tárolt programot.

---

[14] Egyetlen korlát, hogy csak akkor hajthat végre egy programot, ha legalább egy x bit van az adott program védelmi szavában.

[15] A tulajdonos pl. megtilthat mindent, még saját magát is kizárhatja a hozzáférésből, de ez a su-ra nem vonatkozik.

A katalógusokra vonatkozó **keresési** jog azt jelzi, hogy van-e jogunk a katalógusbejegyzéseinek eléréséhez. Ha a felhasználó csak olvasási joggal rendelkezik egy katalógusra, akkor a tartalmát ki tudja listáztatni (vagyis meg tudja nézni, hogy egyáltalán milyen nevű bejegyzések találhatók benne), de a katalógusbejegyzéseiről semmilyen információt nem tud kérni, valamint (módjuktól függetlenül) nem tudja azokat elolvasni, futtatni, illetve nem tud írni beléjük. A csak keresési joggal rendelkező felhasználó elérheti a katalógusbejegyzéseit, vagyis (módjuktól függően) olvashatja, végrehajthatja azokat, írhat beléjük, de nem tudja kilistáztatni a katalógus tartalmát (vagyis nem tudja megnézni, hogy milyen bejegyzéseket tartalmaz). Ezért általában a katalógusoknál célszerű a két jogot együtt állítani.

A programok egy része állományokba ír, új állományokat hoz létre, vagy egyéb olyan tevékenységet végez, ami nincs minden felhasználó számára engedélyezve. Ha mégis azt akarjuk elérni, hogy a programot olyan személy is használhassa, akinek egyébként nincs elérési joga a program által használt állományokhoz (pl. egy kvíz program adatbázisához), különleges módot kell állítani a programállományra. Az ún. **setuid** mód lehetővé teszi, hogy a rendszer a programot futtató felhasználót saját uid-je helyett a programállomány tulajdonosának uid-jével azonosítsa (természetesen csak a program futásának időtartamára). Így a felhasználó által futtatott program mindazt végrehajthatja, amit a programállomány tulajdonosa végrehajthat. A **setgid** mód teljesen azonos a setuid móddal: a rendszer a felhasználó valódi gid-je helyett a programállomány tulajdonosának gid-jét (gid-jeit) használja a hozzáférési jogok megállapításánál a program futásának időtartama alatt.

Azok a programok, amelyek állományokat hoznak létre, a létrehozáskor beállítják a létrehozandó állomány védelmi kódját. Így történik ez akkor is, ha pl. editorral létrehozunk egy új állomány. Felmerül a kérdés, hogy mit állítson be egy ilyen általános célú program. Legyen szigorú, és tiltson meg mindent? Vagy legyen elnéző és engedélyezzen mindent? A megoldás egyszerű: adja meg a felhasználó, hogy mit akar. Az egységes kezelés érdekében ezt a funkciót, pontosabban a probléma megoldását támogató funkciót beépítették a rendszerbe. Ez az **umask,** ami egy olyan egész érték, amelyet a rendszer minden új állomány létrehozásakor figyelembe vesz. Mégpedig úgy, hogy azokat a védelmi biteket (csak az alsó 9 bitet), amelyek értéke 1 az umask-ban, a létrehozandó állomány védelmi kódjából kitörli. Mivel az umask egy olyan folyamathoz tartozó adat, amely örökölhető, elegendő a fő shell umask értéket beállítani, és így minden, az ettől a shell-től közvetlenül vagy közvetve származó folyamat megkapja. (Az öröklést ld. a folyamatokról szóló fejezetben.) Így a felhasználó ízlésére van bízva, hogy milyen umask értéket használ, és ennek megfelelően milyen védelemmel jön létre a példában szereplő editorral létrehozott állomány.

A fent ismertetett védelmi rendszer működéséből logikusan következik az az első hallásra furcsa tény, hogy egy állomány törléséhez nem szükséges az, hogy magához az állományhoz bármilyen joggal is rendelkezzünk. Az állományt csak akkor törölhetjük, ha az azt tartalmazó katalógushoz van írási jogunk. Ez logikus,

hiszen az állomány törlése nem más, mint a megfelelő bejegyzés törlése a katalógusból.

3.3 Védelmi kódok jelölése

Az "ls -l" paranccsal[16] a katalógus tartalmáról, illetve bejegyzésekről készített lista tartalmazza a bejegyzések védelmi kódját is. A védelmi kódot egy 10 karakter hosszú sorozat jelzi. Ennek felépítése a következő:

drwxrwxrwx

Ahol az első karakter[17]:

- **b** ha a bejegyzés egy blokk-orientált speciális állomány,
- **c** ha a bejegyzés egy karakter-orientált speciális állomány,
- **d** ha a bejegyzés egy katalógus,
- **l** ha a bejegyzés egy szimbolikus link.
- **-** ha a bejegyzés egy egyszerű állomány,

A következő 9 karakter hármas csoportokra osztható. Az első csoport a tulajdonos, a második a csoporttárs, a harmadik hármas pedig bárki más számára engedélyezett jogokat tükrözi. A hármas csoporton belül a három karakter sorrendben a következő lehet: "**r**", "**w**" és "**x**". ezek bármelyike helyén "**-**" is állhat. Jelentésük pedig rendre az olvasási (**r**ead), írási (**w**rite), végrehajtási (e**x**ecution) jogot vagy "-" esetén annak hiányát jelenti.

A tulajdonos lehetőségeit ismertető csoportban "**x**" helyén "**s**" áll, ha az állomány setuid módban van. Hasonlóképpen a csoporttárs lehetőségeit ismertető csoportban "**x**" helyén "**s**" áll, ha a bejegyzés setgid módú.

3.4 Kapcsolódó parancsok

A védelmi rendszerrel kapcsolatos fontosabb parancsok, melyeknek felhasználói leírását a függelék tartalmazza: (A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai.)

chmod(1)	-	védelmi kód módosítása
chown(1)	-	tulajdonos módosítása
chgrp(1)	-	csoport módosítása
ls(1)	-	attribútumok kiíratása
passwd(1)	-	jelszó változtatása
umask	-	umask állítás (beépített shell parancs)

[16] Minden olyan parancs, amely a védelmi kódokat megadja, az itt leírt jelölési módot alkalmazza.

[17] A teljesség kedvéért meg kell említeni, hogy néhány speciális esetben más karakter is lehet az első pozíción, de ezeket ebben a leírásban nem tárgyaljuk.

4. Felhasználói felület, shell-ek

A UNIX rendszer egyik érdekessége, hogy a felhasználói felületet megvalósító parancsértelmező (shell) egy közönséges felhasználói programként van megvalósítva. Ez a parancsértelmező nem integráns része az operációs rendszernek és nem élvez speciális megkülönböztetéseket. Így ezt bárki lecserélheti a saját felhasználói programjára. A rendszerbe való bejelentkezéskor a rendszer a hívónév (user azonosító) és a jelszó alapján ellenőrzi a felhasználó belépési jogosultságát, majd a password állomány a hívónévhez tartozó sorának megfelelő mezőjében megadott programot elindítja. Ez rendszerint valamelyik shell (sh, csh, ksh, tcsh stb.), de lehet valami más program is. Mindegyik felhasználónak joga van ezt a programot lecserélni. Erre a legtöbb rendszerben a **chsh** vagy a **passwd** parancs szolgál. A módszer rugalmasságát mi sem bizonyítja jobban, hogy több különböző shell terjedt el, és van ma is használatban. A legismertebbek:

sh Bourne shell, ami szinte minden UNIX változatban megtalálható, és mindenki alap shell-nek tekinti. Ismerete azért is fontos, mert igen sok installáló és egyéb segédfunkciót ellátó ún. shell script készült ezzel a változattal.

ksh Korn shell, amely a Bourne shell és a C shell egyfajta ötvözete.

csh C shell, ami sokban hasonlít a C programozási nyelvhez.

tcsh A C shell kibővített változata.

vsh Visual shell, amely menüvezérelt felületével megpróbált alkalmazkodni a képernyős megjelenítőkhöz.

bash Bourne again shell, amely a Bourne shell kibővített változata. Ez lehetővé teszi a parancssor javítását is a PC-s környezetben megszokott módon.

Az egyes shell változatok között a különböző kényelmi szolgáltatásokban, valamint a programozói felületükben van eltérés. Ugyanis a UNIX shell-ek nem csupán parancsértelmezők, hanem programozási nyelvek is egyben.

A UNIX shell-ek legfontosabb tulajdonságai:

- Parancsértelmező, amely a parancsokat a standard bemenetről vagy állományból olvassa.
- Egyszerűen kialakíthatók ún. shell script-ek vagy parancsállományok, amivel a parancskészlet bővíthető.
- A shell script-ek paraméterezhetősége megegyezik a programok paraméterezhetőségével.
- Programnyelv, amely string változókra és hatékony vezérlési szerkezetekre épül. A C nyelv függvényéhez hasonló függvények használata lehetséges.
- Egyszerű szintaxissal lehetővé teszi a B/K átirányítást, a csővezeték szervezést, valamint a processzek áttekinthető kezelését.
- Egyszerűen konfigurálható a felhasználó igénye szerint.

A következőkben áttekintjük a UNIX shell-ek általános működési mechanizmusát és tulajdonságait egy konkrét példán keresztül. Ez a konkrét példa a Bourne shell (sh), amely nevét programozójáról Stephen R. Bourne-ről kapta.

4.1 Parancssor értelmezése

A UNIX shell lényegében egy parancsértelmező program, amely beolvassa a felhasználó által begépelt sorokat, és azt más programok végrehajtását előíró kérésekként értelmezi. A shell által elfogadott nyelv nem bonyolult, az egyes parancsok kezelése pedig egészen egyszerű és szabályos. Egy parancssor a legegyszerűbb formában a parancs nevét tartalmazza, amit szükség esetén a parancsnak szóló argumentumok követnek szóközökkel elválasztva. pl:

```
parancs arg1 arg2 ... argn
```

A parancsértelmező különálló karaktersorozatokká bontja fel a parancs nevét és argumentumait. Ezután egy beállítható út (PATH) mentén haladva a különböző katalógusokban megkeresi a parancs nevű állományt. Ha sikerült megtalálni, akkor azt végrehajtja. Az argumentumként megadott szavak a parancs számára hozzáférhetővé válnak. Amikor a parancs végrehajtása befejeződött, a shell visszatér a saját végrehajtási állapotába, és egy prompt string kiírásával jelzi, hogy kész egy újabb parancs fogadására. A parancssorban szereplő állománynév után következő argumentumok a következő négy kategória valamelyikébe esnek:

- Egyszerű karakterlánc.
- Állománynév, amit "<",">" vagy ">>" előz meg.
- Egy karakterlánc, amely állománynév helyettesítő karaktert is tartalmaz.
- "` `" jelek közé zárt újabb parancs, ami ún. parancshelyettesítést eredményez.

4.2 Egyszerű argumentumok

Az egyszerű argumentumok a parancs számára stringek tömbjeként adódnak át, amit az illető program értelmez. Az a tény, hogy a parancsértelmezőn keresztül az argumentumok külön-külön stringekként adódnak tovább, lehetőséget teremt az argumentumok egységes kezelésére. Jól ismerjük azokat a rendszereket, amelyekben a különböző parancsok argumentumait hol vesszők, hol pontosvesszők, hol zárójelek választják el egymástól és hogy mikor mire van szükség, annak ismeretéhez jó emlékezőtehetség vagy egy állandóan kéznél levő leírás szükséges.

4.3 Bevitel/kivitel átirányítás

A shell által végrehajtott programok eleve három megnyitott állománnyal indulnak. Ezek a 0, 1 és 2 állományleírókhoz vannak hozzárendelve. Amikor egy ilyen program végrehajtása megkezdődik, az 1-es állományleíró írásra van megnyitva, és ez lesz alapértelmezés szerint a standard kimenet. Az alább ismertetendő kivételektől eltekintve ez a felhasználó terminájához van rendelve. Így azok a programok, amelyek a kezelővel közölni akarnak valamit, rendszerint az 1-es állományleírót használják. A 0-ás állományleírójú állomány viszont induláskor olvasásra van megnyitva, és azok a programok, amelyek a kezelő üzeneteit akarják beolvasni, ezt az állományt olvassák. A felhasználó a shell segítségével megváltoztathatja a standard B/K hozzárendeléseket. Ha egy parancs valamelyik argumentuma előtt a ">" jel áll, a parancs végrehajtásának tartama alatt az 1-es

állományleíró a ">" után megnevezett állományra fog vonatkozni. Például az ls parancs közönséges esetben a felhasználói terminálra listázza ki a pillanatnyi katalógusban levő állományok nevét. Az

```
ls  > oda
```

parancs viszont létrehozza az oda nevű állományt, és a listát abba írja. Tehát a "> oda" argumentum jelentése: helyezd az outputot az oda állományba. Hasonlóképpen a standard input is átirányítható a "<" jel segítségével. Így a "< szöveg" argumentum jelentése: vedd az inputot a szöveg állományból. Bár a "<" vagy ">" után következő állománynév a parancsnak szóló argumentumnak látszik, a valóságban azt a shell értelmezi, és egyáltalában nem adódik át a parancsnak, így a programoknak nem kell külön még azzal is törődniük, vajon történt-e átirányítás. A 2-es állományleíró az 1-es állományleíróhoz hasonlóan rendszerint a terminál outputhoz kapcsolódik. Amikor egy ">" jel segítségével eltérítettük az outputot, a 2-es állomány továbbra is a terminálhoz kapcsolódik, így a programok ilyenkor is tudnak diagnosztikai üzeneteket generálni, amelyek nem tűnnek el észrevétlenül az output állományban.

Az állománynév előtt álló ">>" jel azt jelenti, hogy az outputtal nem kell felcserélni az állomány egész tartalmát, hanem az állomány végéhez kell az outputot függeszteni (ha az már létezett).

4.4 Csővezeték

A UNIX rendszer egyik igen hatékony programozási lehetősége a csővezeték (pipe). A csővezeték egy olyan speciális állomány, ami egy FIFO-t (First In First Out) valósít meg. Az érdekesség abban van, hogy ez a FIFO állományként kezelhető: Két állományleíróval hivatkozhatunk rá, egyiken írhatunk bele, a másikon pedig a beírási sorrendben kiolvashatjuk a beírt adatokat[18]. Azaz a csővezetékbe egyik irányból beírt információ a másik irányból kiolvasható, e művelet szinkronizálását, ütemezését, pufferelését a rendszer automatikusan elvégzi. A csővezetékek használata szorosan kapcsolódik a szabványos bevitel/kivitel fogalmához. Miután a csővezeték is ugyanúgy kezelhető, mint bármely más állomány, ezért a standard bemenet ill. kimenet csővezetékbe is irányítható. Így lehetőség van az egyik program standard kimenetét közvetlenül egy másik program standard bemeneteként használni. Ez azért fontos, mert gyakran adódik olyan helyzet, hogy egy program eredménye egy másik program bemenő adata lesz. A közbülső eredmény számunkra érdektelen, csak a második program által adott eredményt hasznosítjuk. Ezt a shell számára szemléletes módon függőleges vonallal ("|") adhatjuk meg. Ekkor a függőleges vonal bal oldalán levő program standard kimenetét csővezetéken keresztül összekapcsolja a jobb oldali program standard bemenetével. Így az

```
ls  | pr | lpr
```

[18] A pipe mérete véges, ha betelik, akkor az író folyamat felfüggesztődik, mindaddig, amíg nem ürül a FIFO.

parancssorban az **ls** kilistázza a pillanatnyi katalógusban levő állományok nevét, a kimenetet csővezetéken keresztül átadja a **pr** programnak, amely oldalakra tördeli és dátumozott fejléccel látja el, majd újabb csővezetéken keresztül a végleges eredményt átadja az **lpr** programnak, ami pedig kinyomtatja az inputként kapott állományt. Természetesen a fenti feladat ideiglenes állományok segítségével is elvégezhető lett volna, de ekkor még az ideiglenesen használt munkaállományok törléséről is nekünk kellene gondoskodnunk.

A csővezeték-szervezés lehetőségeinek előnyei:

- Nincs szükség ideiglenes állományokra, amit később úgy is letörölnénk.
- Mivel a folyamatok "párhuzamosan" futnak, a FIFO-nak nem kell nagynak lenni.
- A pipe használata gyorsabb, mivel az a legtöbb esetben - a rendszer terhelésétől Függően - memóriában keletkező állományként jelenik meg (egy FIFO csupán 8 Kb).
- Az eredmény a cső végén már azelőtt megjelenhet, mielőtt az első program az összes bemenetét feldolgozná.

A B/K átirányítás a csővezetékkel kombinálva egy nagyon hasznos lehetőséget ad a UNIX-ban. Ez nem más, mint a **szűrő** programok lehetősége. Az olyan programokat, amelyek standard bemenetüket (műveletvégzés után) a standard kimenetükre másolják, szűrőknek nevezzük. Szűrő programokkal igen sokfajta utófeldolgozást tudunk végezni egy program eredményén, anélkül, hogy az eredeti programot megváltoztatnánk. A UNIX-ban több mint száz ilyen egyébként igen egyszerű feladatot ellátó szűrő van, amelyekkel igen bonyolult feladatokat is megoldhatunk. Hasznos szűrők például azok a programok, amelyek karakter-átkódolást, sorok minta szerinti kiválasztását, rendezését, titkosítást vagy titkosított szöveg visszakódolását végzik. Ha megvizsgáljuk, a UNIX segédprogramok (utility) nagy része szűrőként is képes működni. Általános, hogy ha egy program nem kap bemeneti állományparamétert, vagy ha az "-", akkor a standard bemenetről olvas.

Programok szűrőként való használatának lehetősége alapvetően befolyásolja azt a módot, ahogyan egy adott feladat megoldható, különösen ami a szövegfeldolgozást illeti. Vegyük például azt a három, létező programot, amelyek karakterek átkódolását (tr), ismétlődő sorok elhagyásával történő rendezést (sort) és két rendezett állomány olyan összehasonlítását végzik (comm), ahol az első állományban szereplő, de a másodikban nem szereplő sorok kerülnek megjelenítésre. Ezeket a programokat az on-line szótárral kombinálva olyan "pipeline"-t (parancsfolyamot) kapunk, amely kiírja egy dokumentumból mindazokat a szavakat, amelyek nem találhatók meg a szótárban. Más szóval kiírja a valószínű helyesírási hibákat.

4.5 Állománynév-helyettesítés

A parancs argumentumában megadott "*****", "**?**", és "**[**...**]**" jelek ún. állománynév-helyettesítő jelek. Ezek jelentése:

*****	Nulla vagy tetszőleges számú tetszőleges karakter.
?	Pontosan egy tetszőleges karakter.

[abc] Az "**a**" vagy "**b**" vagy "**c**" karakter egyike.
[m-n] **m-n** intervallumból egy karakter.

Az olyan argumentumok, amelyek állománynév-helyettesítő karaktereket tartalmaznak, állománynevekből álló egyszerű argumentumok sorozatára cserélődnek le. Például "*f"-ből azoknak az aktuális katalógusban található állományneveknek a sorozata lesz, amelyek az "f" karakterekre végződnek. A helyettesítő karakterek közül több is szerepelhet egyidejűleg ugyanabban az argumentumban. Pl: az "[a-z]*" a kisbetűvel kezdődő neveket jelenti.

Az a tény, hogy a kifejtő mechanizmust a shell tartalmazza, több előnnyel jár:

- A kifejtést megvalósító program csak egyszer szerepel a rendszerben (helytakarékosság).
- A programoknak a kifejtéssel nem kell foglalkozniuk.
- A kifejtő algoritmus alkalmazása így bizonyosan mindig egységes lesz.

4.6 Parancshelyettesítés

Igen hasznos és érdekes szolgáltatása a UNIX shell-eknek az ún. parancshelyettesítés, ami lehetővé teszi, hogy egy programnak paramétereként egy másik program eredményét adjuk át. Ilyenkor a megfelelő parancsargumentumot "`" (visszafelé dőlő aposztrof) párba kell zárni. A

```
more `grep -l cicuska *.txt`
```

parancs hatására a more program kiírja az összes olyan állományt az aktuális katalógusból, amelynek neve .txt-re végződik, és tartalmazza a cicuska mintát. A működés egyszerű: A grep megkeresi, a megadott (*.txt) állományokban a cicuska mintát, és kiírja a megfelelő állományok nevét a standard kimenetre, amit a more program paraméterként kap meg. (Képzeljük el, hogy a fenti feladatot hogyan oldottuk volna meg parancshelyettesítés nélkül!)

4.7 Parancssorozatok

Ha a parancsokat egymás után pontosvesszővel elválasztva írjuk, akkor úgynevezett parancssorozathoz jutunk. Így pl. az

```
ls ; who
```

parancs először kilistázza a pillanatnyi katalógust, majd közli, hogy kik vannak bejelentkezve a rendszerbe. Parancssorozatot nem csak pontosvessző segítségével alkothatunk, hanem a '||' és a '**&&**' jelekkel is. Ekkor egy feltételesen végrehajtódó sorozathoz jutunk. Minden parancs egy úgynevezett megállási értéket (exit status) hagy maga után. Gyakorlati szabály, hogy ha egy parancs hibamentesen fejeződött be, akkor ez a megállási érték 0, egyébként nullától különböző. Ezt feltételként kezelve feltételesen végrehajtódó utasításokat készíthetünk. A '||' és az '&&' jelek jelentése azonos a C nyelvben megszokottal. Ennek megfelelően pl. a

```
parancs1 || parancs2
```

```
parancs3 && parancs4
```

sorozatból a parancs2 csak akkor fog végrehajtódni, ha a parancs1 hamis (azaz nem nulla) megállási státusszal állt meg. A parancs4 pedig csak akkor fog végrehajtódni, ha a parancs3 igaz (azaz nulla) megállási státusszal állt meg.

4.8 Szinkron és aszinkron folyamatok

Az eddigi példákban az egymás utáni parancsok egymást követve szinkron módon hajtódtak végre. Lehetőség van azonban aszinkron végrehajtásra is. Ha egy parancsot az "**&**" jel követ, akkor a shell nem várja meg a parancs befejeződését, hanem a prompt jel kiadásával újabb parancsra várva azonnal visszajelentkezik, miközben a kiadott parancs aszinkron módon a shell-től leválasztva fut tovább. Például a

```
cc forrás > üzenet &
```

parancs hatására megtörténik a forrásállomány lefordítása, a standard kimenet az üzenet állományba íródik. Függetlenül attól, hogy milyen sokáig tart a fordító működése, a shell azonnal visszatér, és újabb parancsra vár. Amikor a shell nem várja meg egy parancs végrehajtásának a befejezését, leválasztott, vagy aszinkron végrehajtásról beszélünk. Ilyen esetben kiíródik az illető parancsot futtató folyamat azonosítójának száma. Ennek az azonosítónak a segítségével tudunk később várakozni a parancs befejeződésére, vagy ezt használva tudjuk a végrehajtást félbeszakítani. A fenti a példában a parancs kimeneteként nem a terminált használtuk, mert a különböző parancsok által kiadott output a terminálon összekeveredett volna.

A fentiekben ismertetett parancs-végrehajtási műveletek zárójelezhetők. Például:

```
(date; ls) > lista &
```

parancs a lista nevű állományra kiírja az aktuális dátumot, majd az aktuális katalógus tartalmát. E parancs kiadása után a shell azonnal visszatér és újabb parancsra vár.

4.9 Parancsállományok

Mint láttuk, a shell maga is egy program, amely rekurzív módon hívható. Tegyük fel, hogy a vezér nevű állomány az alábbi sorokat tartalmazza:

```
cc forrás
mv a.out prog
prog
```

Az mv parancs hatására az a.out[19] állomány új neve prog lesz. Ha tehát valaki a fenti 3 sort begépeli a terminálon, a forrás nevű állomány lefordítódik, az így

[19] Az a.out a C fordító bináris outputja, amely programként végrehajtható.

kapott program prog-ra neveződik át, majd a prog végrehajtódik. Ha ezek a sorok a vezér nevű állományban foglalnak helyet, az

```
sh vezér
```

parancs hatására a shell a fenti parancsokat egymás után végrehajtja. Ha a vezér nevű állomány rendelkezik végrehajtható védelmi kóddal, akkor elegendő az állomány nevét leírni, mert ilyenkor az éppen futó shell a parancsállomány végrehajthatóságát felismerve a fenti módon elindít egy újabb shell-t, ami végrehajtja az állományba beírt parancsokat. A beállított végrehajtási módú parancsállományokat a segédprogramokhoz hasonló módon lehet használni. Ez ugyanannak a parancsszintaxisnak a használatát engedi meg, függetlenül attól, hogy az állomány egy bináris, végrehajtható programot vagy futtatható parancsok szöveges specifikációját tartalmazza. Ugyancsak megengedett parancsállományokon belül programok és újabb parancsállományok keveredése, tetszőleges mélységben skatulyázva.

Mint már korábban említettük, a shell nem csupán egy egyszerű parancsértelmező. A shell igen széleskörű képességekkel rendelkezik:

- Tetszőleges információfolyam (például diagnosztikai üzenetek) átirányítása.
- Kimenet hozzáfűzése már meglévő állományokhoz.
- Csővezetékek szervezése.
- Állománynevek helyettesítése.
- Parancshelyettesítés.
- Programok végrehajtása más programok végrehajtásától függő feltételek szerint.
- Iteratív programvégrehajtás (for/while ciklusok).
- Parancsnyelvi argumentumok használata.
- Shell változók használata.

Ezek a képességek együttesen egy olyan parancsértelmezővé teszik a shell-t, amellyel komplex feldolgozási feladatok végrehajtását lehet előírni. Mivel az egyes shell változatok némileg eltérnek a bonyolultabb parancsok használatában, ezért ezeket a megfelelő shell változatok leírásánál a mellékletben mutatjuk be.

4.10 Kapcsolódó parancsok

A shell-ekhez kapcsolódó fontosabb parancsok: (A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai.)

basename(1) - eltávolítja az állománynévből az útnevet és a kiterjesztést
bash(1) - Bourne again shell
chsh(1) - login shell változtatása
csh(1) - C shell
dirname(1) - csak a katalógusnevet írja ki az állománynévből
ksh(1) - Korn shell
sh(1) - Bourne shell
tcsh(1) - Módosított C shell
tee(1) - standard kimenet másolása

test(1) - feltétel kiértékelése
vsh(1) - visual shell
wait(1) - várakozás aszinkron folyamatra

5. Folyamatok

A UNIX rendszer egy a memóriába betöltött működő programot a programváltozókkal és a programhoz tartozó környezeti változókkal együttesen folyamatnak (process) nevezi. Érdekessége a UNIX rendszernek, hogy a folyamatok is egy fa szerű hierarchiába rendezhetők. Minden folyamatnak van ugyanis egy szülőfolyamata, amelytől a gyerek különböző tulajdonságokat örököl. Egy folyamat a fork rendszerhívással hozhat létre új folyamatot. Ennek során az új folyamat örökli a szülőfolyamat

- megnyitott állományait, és azok leíróit,
- szignálok kezelésére vonatkozó előírásait,
- az umask és ulimit értékeit,
- a teljes memóriaképét,
- a folyamat gépi regisztereit.

Az ily módon létrejövő gyerekfolyamat a fenti információk birtokában dönthet, mit használ fel, és lehetősége van egy teljesen új folyamat létrehozására is. Ezt az exec rendszerhívással hajthatja végre, amely során az új folyamat örökli az eredeti folyamat

- megnyitott állományait, és azok leíróit,
- szignálok kezelésére vonatkozó egyes előírásait,
- az umask és ulimit értékeit.

Természetesen van egy ősfolyamat (init), amitől a rendszer indulásakor a folyamatok származnak. Ennek a folyamatnak igen fontos szerepe van, hiszen egy felhasználó kijelentkezésekor ez a folyamat hozza létre a következő felhasználót beléptetni képes folyamatot (getty). Más fontos szerepe is van az ősfolyamatnak, mégpedig az, hogy átvegye a szülő (mostohaszülő) szerepét, ha egy gyerekfolyamat szülője megszűnik.

A UNIX ütemezője ún. prioritásos ütemező, bár a prioritás értékeket többnyire belső szabályok határozzák meg, és csupán kis mértékben szabályozhatja maga a folyamat.

5.1 Kapcsolódó parancsok

A folyamatokkal kapcsolatos fontosabb parancsok, melyeknek felhasználói leírását a függelék tartalmazza: (A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai.)

kill(1) - szignál küldése a folyamatnak
nice(1) - prioritás állítása
nohup(1) - hangup szignál figyelmen kívül hagyása
ps(1) - folyamatállapot kiíratása

6. Fontosabb segédprogramok és szűrők

Mint azt a csővezetékekről szóló fejezetben leírtuk, a UNIX rendszer számtalan egyszerű segédprogramot tartalmaz, amelyek többsége szűrőként is képes működni. Ezek a programok többnyire igen egyszerű feladatot látnak el, ugyanakkor a mindennapi programozói tapasztalat azt mutatja, hogy éppen ezekre van a leggyakrabban szükség programfejlesztés, hibakeresés, vagy éppen dokumentációírás közben. Az is előfordulhat, hogy ezen programok többségét egy programozó a munkája során már többször megírta, és órákat, esetleg napokat töltött ezekhez hasonló feladatokat ellátó programocskák belövésével. Erre a UNIX-ban nincs szükség, mivel kész, kipróbált programok állnak rendelkezésre. Ezek közül könnyen kiválasztható a legalkalmasabb, hiszen rendelkezésre áll az egészen egyszerű másoló programtól (cat) a programozható riport generátorig (awk) ill. compiler generáló programig (yacc, lex) szinte minden eszköz. Ebben a fejezetben ezen eszközök közül mutatunk be néhányat.

Szinte a legegyszerűbb, de gyakran előforduló feladat szöveges állományok egymáshoz másolása, rendezése, keresés valamilyen minta alapján, egyszerű helyettesítés stb. Ezen feladatokra az M. függelékben ismertetett cat, sort, tr, diff, grep, wc, pr, dd, find programok a legalkalmasabbak. A bonyolultabb feladatokhoz komplexebb programok mint pl. sed, awk, perl, lex, yacc stb. alkalmazhatók. Ezen programokban a keresett mintát egységesen ún. reguláris kifejezés formájában kell megadni.

6.1 Reguláris kifejezések

A legtöbb segédprogram valamilyen szöveges bemenetet dolgoz fel. Ennek feldolgozása során igen sokszor mintát kell illeszteni. Ezt a feladatot úgy próbálták egységesíteni, hogy többnyire egységes szintaxist alkalmaznak a minta megadására. Ez a szintaxis az ún. **reguláris** vagy szabályos kifejezések szintaxisa. Egy szövegmintában általában minden karakter önmagát jelenti, azaz a mintában az adott helyen elő kell fordulnia, de van néhány speciális jel, ami valami mást jelent. Ennek alapján egy mintában a következő karakterek fordulhatnak elő:

c	Maga a c karakter, amennyiben c nem speciális karakter.
\c	A c karakter, kivéve: újsorjel, számjegy, "(" vagy ")".
^	Sor eleje.
$	Sor vége.
.	Egy db akármilyen karakter (kivéve az újsorjel).
[abc]	Egy karakter az "abc"-ből.
[^abc]	Bármely karakter, amely nem "a", "b", vagy "c"
r*	r reguláris kifejezés 0 vagy tetszőleges számszor.
r+	r reguláris kifejezés 1-szer vagy sokszor.*
r?	r reguláris kifejezés 0-szor vagy 1-szer.*

* Csak egrep, awk, lex, perl esetében.

r1r2	r1 és r2 reguláris kifejezés egymás után úgy, hogy r1 a lehető leghosszabban illeszkedjék.
r1|r2	r1 vagy r2 reguláris kifejezés.*
\(r\)	r reguláris kifejezés önmaga, de erre később hivatkozni lehet a \n alakban, ahol n egy számjegy.**
\n	Hivatkozás az n. \(r\) alakú reguláris kifejezésre, ahol n egy számjegy. (Az első ilyenre a \1 hivatkozik).**
(...)	Egymásba ágyazott kifejezések.*
üres	Ha a reguláris kifejezés helyén nem áll semmi, akkor az a legutoljára alkalmazott kifejezést jelenti.**

Példák:

1. Az **alma** reguláris kifejezés azt jelenti, hogy az **alma** minta a soron belül bárhol előfordulhat.
2. A **^alma** előírja, hogy az **alma** mintának a sor elején kell előfordulnia.
3. A **^[mh]?alma** kifejezés azt jelenti, hogy sor elején **alma** vagy **malma** vagy **halma** mintának kell előfordulnia.
4. A **^[^mh]alma** kifejezés azokra a sorokra illeszkedik, amelyen nem **malma**, vagy **halma** sorozattal kezdődnek.
5. A **^\([abc]\)\1** kifejezés azokra a sorokra illeszkedik, amelyek vagy **aa**, **bb** vagy **cc** kezdetűek.

6.2 Awk riport generátor

Az **awk**-t sokszor programozási nyelvkény is szokták emlegetni, ennek ellenére elsősorban nem programfejlesztő eszköz, hanem egy általános riport generáló nyelv, amely szöveges inputot tud feldolgozni. Segítségével igen gyorsan és hatékonyan elvégezhetők pl. adatelőkészítési feladatok, vagy egyszerűbb utófeldolgozási munkák is mint pl. naplóállomány feldolgozása. Szintaktikája nagyon hasonlít a C nyelvhez. Az awk soronként olvassa az input-ot, és megpróbálja az egyes sorokat, vagy azok részeit a megadott mintákra illeszteni. Egy awk program általános szerkezete a következő:

```
BEGIN   { kezdeti_tevékenység }
minta_1  { tevékenység_1 }
minta_2  { tevékenység_2 }
          ...
minta_n  { tevékenység_n }
END      { utolsó_tevékenység }
```

A bemenetről beolvasott sorokat az awk az **FS** (field separator) változóban megadott karakter[20] mentén darabokra töri. (Alapértelmezés szerint ez a space és a tab.) Ezen darabokra mind a mintában mind a tevékenységekhez tartozó utasításokban **$1**, **$2**, .. néven ill. a teljes sorra **$0** néven lehet hivatkozni. Ha a bemenetről beolvasott sor illeszkedik valamelyik mintára, akkor végrehajtódik a

** Csak ed, sed és perl esetében.

[20] Egyes programváltozatoknál az FS reguláris kifejezés is lehet.

hozzátartozó tevékenység. Ezután a mintaillesztés folyatódik a következő mintával. Speciális esetben a minta elhagyható, vagy két minta is megadható vesszővel elválasztva. Ha a minta hiányzik, akkor a hozzátartozó tevékenység minden sorra végrehajtódik. Ha viszont két minta van, akkor a hozzátartozó tevékenység minden olyan sorra végrehajtódik, ami az első mintára illeszkedő sor után de a második mintára illeszkedő sor előtt van.

A **minta** legegyszerűbb formája a perjelek közé zárt reguláris kifejezés (/reg.kif/). Ekkor az adott reguláris kifejezést a teljes bemeneti sorra (**$0**) próbálja illeszteni az awk. Lehetőség van azonban tetszőleges összetett feltétel megfogalmazására is. A **BEGIN**-hez ill. **END**-hez tartozó tevékenység az első file feldolgozása előtt ill. az utolsó file feldolgozása után hajtódik végre. Ezek bármelyike elmaradhat.

A **tevékenység** C nyelvhez hasonló utasításokat tartalmazhat. Ha a tevékenység elmarad, akkor a beolvasott sor változtatás nélkül kiíródik a standard outputra. Minden a C nyelven megszokott operátor ill. több fajta vezérlési szerkezet is alkalmazható az utasítások kialakítására. Az utasítások egy része **változókon** operál. Érdekessége, és fontos tulajdonsága a nyelvnek, hogy a váltók egyszerre string ill. numerikus típusúak. Azaz mindig az éppen megfelelő alakjukban helyettesíti az awk. A változókat nem kell definiálni. Kezdeti értékük az üres string ill. a nulla szám. Ilyen string ill. numerikus típusú változókból tömböt is képezhetünk, amit szintén numerikus vagy string értékkel indexelhetünk. A tömbök ilyen megvalósítása egy asszociatív tárral kapcsolódik össze, ami igen hasznos sok esetben.

Utasítások:

if (feltétel) utasítás [**else** utasítás]
while (feltétel) utasítás
do utasítás **while** (feltétel)
for (kifejezés; feltétel; kifejezés) utasítás
for (változó **in** tömb) utasítás
break
continue
next # a maradék minták átlépése
exit [kifejezés] # megállás visszatérési értékkel
return [kifejezés][21]
delete tömb[index]
kifejezés # pl: változó = kifejezés
{ [utasítás] ... }
print [kifejezés_lista] [>kifejezés]
printf formátum [, kifejezés-lista] [>kifejezés]
függvény_hívás

A **kifejezések** a C nyelvben megszokott **operátorok** segítségével alkothatók (+, -, *, /, %, ++, --, += -= = stb.). Speciális operátor a konkatenáció, amit szóközzel kel jelölni. A logikai operátorok ill. a relációs operátorok is a C nyelvnek megfelelőek (&&, ||, !, <, <=, ==, > >= !=). Új operátor az ún. illesztő operátor (~, !~), amellyel

[21] Csak az újabb awk változatokban.

egy kifejezés és egy reguláris kifejezés "hasonlítható" össze. A C nyelvhez hasonlóan tömb indexelésére a szögletes zárójeleket kell használni. A string-konstansokat idézőjelek közé ("string"), a reguláris kifejezéseket pedig perjelek közé kell zárni.

A **feltételek** formája:

kifejezés illesztő_operátor reguláris_kifejezés
kifejezés1 reláció_operátor kifejezés2
kifejezés in tömb_változó [21]
(kif1, kif2, ...) in tömb_változó [21]

ahol az illesztő operátor a ~ (illeszkedik) vagy a **!**~ (nem illeszkedik) lehet. A relációs operátor a C nyelv bármelyik relációs operátora lehet.

Beépített függvények:

exp(x)	e x. hatványa
log(x)	ln(x)
atan2(x)	arcus_tangens(x)
cos(x)	cos(x)
sin(x)	sin(x)
sqrt(x)	négyzetgyök x
rand	véletlenszám 0 és 1 között
srand[(x)]	véletlenszámgenerátor indítás x-től
int(x)	x egészrészre csonkolása
sprintf(form, expr)	a formázott stringgel tér vissza
length[(arg)]	a sor ill. az arg hosszát adja
substr(s,m,n)	s string m. karakterétől n karakter
index(s1,s2)	s1-ben s2 első előfordulásának helye
sub(s1,s2,s3) [21]	ed s parancsa az s3 stringre: s/s1/s2/
gsub(s1,s2,s3) [21]	ed s parancsa az s3 stringre: s/s1/s2/g
match(s,re) [21]	s stringben re reguláris kif.-t keresi
split(s,a,fs) [21]	s-t fs szeparátor mentén darabolja
close(file)	file lezárása
getline	$0-ba olvassa a következő input sort
getline <file	a getline a file-ból olvas
getline var	$0 helyett var-ba teszi
getline var <file	
cmd | getline	cmd kimenetét a getline-ra pipe-olja
system(cmd) [21]	cmd-t végrehajtja

A printf és sprintf formátumlistája a C nyelv printf függvényével egyező. A standard output a >, >> ill. | jellel átirányítható.

Felhasználó által **definiálható függvények**:

function név(args,...) { utasítások }

Ha az átadott változó egyszerű változó, akkor értékparaméterként adódik át, ha viszont tömb, akkor referenciaként.

Speciális változók:

FS	mező szeparátor
NF	mezők száma a beolvasott sorban
NR	olvasott rekordok száma
OFS	output mező szeparátor
ORS	output rekord szeparátor
FILENAME	aktuális input file neve
ARGC	parancsnyelvi paraméterek száma
ARGV	parancsnyelvi paraméterek
FNR	olvasott rekordok száma az aktuális file-ban
OFMT	output számformátum
RS	input record szeparátor

Az awk a feldolgozást vezérlő programot indítási paraméterként kapja, vagy egy állományból olvassa be (-f kapcsoló). A feldolgozandó szöveget pedig a standard bemenetről, vagy paraméterként kapott állományokból veszi (ld. M függelék).

Példák:

1. Írjuk ki a standard inputról érkező 72 karakternél hosszabb sorokat:
```
awk 'length > 72'
```

2. A password file-ból írjuk ki az azonos loginnevű felhasználókat. Ezután írjuk ki a loginneveket a hozzátartozó uid-dal. A feladatot megoldó awk program:
```
BEGIN {
     FS = ":"
}
{
    if (user[$1]) {
        printf "%s duplicated\n", $1
    }
    user[$1] = $3
}
END {
     for (i in user)
         printf "%s = %s\n", i, user[i]
}
```
Az awk indítása:
```
awk -f awk_program /etc/passwd
```

3. Fordított sorrendben írjuk ki egy file mezőit:
```
{ for (i = NF; i > 0; --i) print $i }
```

4. Írjuk ki az összes sort a start, és a stop sorpárok között:
```
awk '/start/,/stop/'
```

6.3 Perl riport generáló nyelv

A **perl** (Practical Extraction and Report Language) az awk-hoz hasonlóan egy általános riport generáló nyelv, amely nem csak szöveges inputot tud feldolgozni.

A nyelvet azzal a céllal definiálták, hogy egyesítse az **sh**, **awk**, és a **C** nyelv előnyeit. Így segítségével nem csak az awk-val elvégezhető egyszerű szövegfeldolgozási feladatok végezhetők el, hanem bonyolultabb programozási feladatok is. Előszerettel használják a rendszer-adminisztrátorok, mivel igen hatékonyan megfogalmazhatók segítségével összetet rendszer-adminisztrációs feladatok is. Szintaktikája nagyon hasonlít a C nyelvhez, de az előbbiekből adódóan az sh-hoz is. Első ránézésre egy perl program elég rémisztő, mivel más nyelvektől eltérően rengeteg speciális jelet alkalmaz, amelyektől nem túlzottan olvasmányos. A pontos szintaxist itt terjedelmi okok miatt nem ismertetjük, csupán egy-két jellemző tulajdonságát emeljünk ki:

- Lista változója is van.
- Az awk-hoz hasonlóan a változók hol string-ek, hol numerikus változók, de a név első karaktere bizonyos nem egyértelmű estekben pontosítja az értelmezést.
- Külön operátorok vannak string-ekre, és a numerikus értékekre.
- Sok unix utility függvényként jelenik meg benne (pl. tr, find, sort).
- A reguláris kifejezéseket nagymértékben kiegészítették a grep-hez képest.
- Sok shell tulajdonság beépült a nyelvbe. Így pl a set is. Pl. a
  ($wday, $day, $month) = `date`
  =~ /^(\w+)\s+(\d+)\s+(\w+)\s+.*/;
  hatására lefut a date parancs, és az eredményt a megadott reguláris kifejezésre illeszti. Ha ez sikerült, akkor a megadott változókba kerülnek a zárójelezett mintarészletek.
- A dbm adatbáziskezelőt beépítetten tudja használni.
- C nyelvben megszokott módon hívhatjuk a renszerhívásokat is. (pl. fork, socket, pipe, shmget (shared memory kezelés) stb.).
- Van objektumorientált kiterjesztése.

A perl a feldolgozást vezérlő programot indítási paraméterként kapja (-e kapcsoló), vagy egy állományból, vagy a standard bemenetről olvassa be. A feldolgozandó szöveget pedig a standard bemenetről, vagy paraméterként kapott állományokból veszi (ld. M függelék).

Példák:

1. Konvertáljuk a bemenetként kapott file-okat ún. HNA-2 ajánlás szerinti ékezetes formáról IBM PC-n használt EXTENDED ASCII kódúvá:

```
#!/local/bin/perl
# Perl program HNA-2 -> EXTENDED ASCII konverzio
#
while (<>) {
    s/a\'/á/g;      s/e\'/é/g;      s/i\'/í/g;      s/o\'/ó/g;
    s/u\'/ú/g;      s/A\'/Á/g;      s/E\'/É/g;      s/I\'/Í/g;
    s/O\'/Ó/g;      s/U\'/Ú/g;      s/o:/ö/g;       s/u:/ü/g;
    s/O:/Ö/g;       s/U:/Ü/g;       s/o\"/ő/g;      s/u\"/ű/g;
    s/O\"/Ő/g;      s/U\"/Ű/g;      s/\\'/\'/g;     s/\\"/\"/g
    s/\\:/:/g;      print;
}
```

2. Dolgozzuk fel a last program által szolgáltatott eredményt:

```
#!/local/bin/perl
```

```
# Perl program a last feldolgozasahoz
#
while (<>) {
    ($name, $term, $addr, $from, $to, $t_hours, $t_mins) =
     /^(\w+)\s+(\w+)\s+([\w\.]*)\s+(\w+ \w+\s+\d+ \d+:\d+) - (\d+:\d+)  \((\d+):(\d+)\).*$/;
#avida     ttyp0    cs.cs.appstate.e Tue Oct 12 01:04 - 01:23  (00:18)

#print "$name $term $addr $from $to $total\n";

   if ($name ne "") {
    $logins{$name} += 1;
    $times{$name} += $t_hours * 60 + $t_mins;
   }
}

print "  NAME   Logins Total time Average time\n";
foreach $i (sort keys(%logins)) {
    $t1 = int($times{$i} / 60);
    $t2 = $times{$i} % 60;
    $t3 = $times{$i} / $logins{$i};
    $t4 = int($t3 / 60);
    $t5 = $t3 % 60;
    printf("%-10s %3d   %3d:%02d     %3d:%02d\n", $i, $logins{$i}, $t1, $t2, $t4, $t5);
}
```

6.4 Lex lexikai analizátor generátor

A lex nyelv szorosan összekapcsolódik a yacc (Yet Another Compiler Compiler) nyelvvel, ami egy fordítóprogram író nyelv. A lex a lexikai analizátor elkészítéséhez nyújt segítséget a yacc pedig a nyelvtan elemzéséhez. Mindkét nyelv C forrásprogramot állit elő, amit utólag a C fordítóval kell lefordítani. A lex nem csak fordítóprogram íráskor lehet segítségünkre, hanem felhasználhatjuk szövegfeldolgozási feladatokra is. A lex program lényegében három fő részből áll:

definíciós rész
%%
lex szabályok
%%
C utasítások

Azokat a sorokat, amelyek nem a sor elején kezdődnek, a lex C utasításoknak tekinti, és átmásolja az eredmény file-ba. Szintén C utasításnak tekinti azokat a sorokat is, amelyek a **%{** karaktereket tartalmazó sor és a **%}** karaktereket tartalmazó sor között helyezkednek el. A **lex szabályok** egy reguláris kifejezésből, és a hozzá tartozó tevékenységből állnak, ami a legegyszerűbb esetben egy vagy több C utasítás. A lex szabályokból a lex fordító egy **yylex** nevű függvényt készít. Ha egy lex által generált C forrást megfelelő módon lefordítunk, akkor futóképes programot kapunk, amely a standard input minden sorát megpróbálja a megadott reguláris kifejezésekre illeszteni. Ha talált illeszkedőt, akkor végrehajtja a megadott tevékenységet (C utasításokat). (Az illesztési sorrend a file-ban megadott sorrendnek felel meg.)

Példák:

1. LF -> CR-LF konverter:

```
%%
\n printf("\r\n");
```

2. Kisbetűkké alakítja a nagybetűket, eldobja a sorvégi szóközöket és minden szóközsorozatot egy szóközzel helyettesít:

```
%%
[A-Z]  putchar(yytext[0]+'a'-'A');
[ ]+$  ;
[ ]+   putchar(' ');
```

3. Lexikai elemeket felismerő program:

```
D       [0-9]
%{
int  comment;
%}
%%
if      printf("IF utasítás\n");
[a-z]+  printf("azonosító: %s\n",yytext);
0{D}+   printf("oktális szám: %s\n",yytext);
{D}+    printf("decimális szám: %s\n",yytext);
"++"    printf("++ op\n");
"+"     printf("+ op\n");
"/*"    {
         do {
           comment = 1;
               while (input() != '*');
               switch (input()){
                 case '/':
               comment = 0;
               break;
                 case '*':
               unput('*');
               }
             } while (comment);
        }
```

6.5 Kapcsolódó parancsok

A fontosabb segédprogramok és szűrők: (A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai.)

awk(1)	-	riport generaló nyelv
cat(1)	-	állományok összefűzése és kiíratása
cmp(1)	-	két állomány összehasonlítása
comm(1)	-	két állomány közös sorainak kiírása
dc(1)	-	asztali számológép
dd(1)	-	állomány konvertálása és másolása
diff(1)	-	szöveges állományok soronkénti összehasonlítása
expr(1)	-	kifejezés kiértékelése
file(1)	-	állomány típusának meghatározása

find(1) - állományok keresése nevük vagy tulajdonságaik alapján
grep, egrep, fgrep(1)
- állományban minta szerinti keresés
lex(1) - lexiakai analizátor generátor
od(1) - oktális, decimális, haxadecimális és ASCII dump
perl(1) - riport generáló nyelv
pr(1) - állományok nyomtatáshoz való előkészítése
sort(1) - állományok sorainak rendezése és egybeolvasztása
split(1) - állomány darabokra tördelése
strings(1) - látható karakterekből álló stringek keresése
tee(1) - standard kimenet másolása
tr(1) - karakter átalakítása
uniq(1) - állomány ismétlődő sorainak kezelése
wc(1) - kiírja a sorok, szavak és karakterek számát

7. Programfejlesztő és karbantartó eszközök

Mint már említettük, a UNIX rendszert eredetileg programfejlesztésre hozták létre. Ezért nem véletlen, hogy sok jól használható programfejlesztő eszközt ill. segédeszközt tartalmaz. Természetesen ezek nem csupán programfejlesztésre alkalmasak, hanem más feladatok is megoldhatók segítségükkel. Ebben a fejezetben ezek közül mutatunk be néhányat.

7.1 Make

Nagyobb programokat rendszerint különböző modulokból modulárisan építünk fel. Az egyes modulok több különböző állományt tartalmazhatnak. A végleges futtatható változatot rendszerint úgy állítjuk elő, hogy az egyes állományokat lefordítjuk, majd összeszerkesztjük. Ez a folyamat több program elindítását igényli különböző paraméterekkel és kapcsolókkal. A gépelési munka és a csoportmunka megkönnyítése érdekében kézenfekvő a megfelelő parancssorozatot egy parancsállományba írni, és azt futtatni. Rendszerint azonban nem kell minden műveletet elvégezni, vagy legalábbis nem minden állományra. Ezen a problémán segít a **make** segédprogram. A make alapötlete egyszerű: összeállítunk egy listát, hogy mely állományok szükségesek egy adott eredményállomány ill. részeredmény-állomány előállításához, és megadjuk, hogy ezek az állományok milyen parancssorozattal hozhatók létre. Ezt a listát egy program segítségével feldolgozzuk, amely a lista és a rendelkezésére álló állományok és azok módosítási ideje alapján képes meghatározni azt a minimális parancssorozatot, amellyel az eredményállomány naprakészen előállítható.

A make számára összeállított lista valójában nem más, mint egy függőségi gráf. A make program pedig értelmezi ezt a gráfot, és újra előállítja a gráf összes olyan elemét, amely nem naprakész, azaz nem létezik, vagy olyan állománytól függ, aminek a módosítási ideje későbbi mint az előállítandó állomány.[22] A make

[22] Ez nem feltétlenül hibátlan módeszer, mert pl. egy hálózaton az idők eltérhetnek egymástól. Azonbam más bonyolultabb módeszerek alkalmazása igen ritkán lenne indokolt.

leírónyelve egy egyszerű makró helyettesítőt is tartalmaz, hogy a függőségeket megadó leírások elkészítése egyszerűbb legyen.

A make leíróállományának szerkezete:

A make leíróállománya (továbbiakban makefile) egyszerű szöveges állomány, amely makró definíciókat, függőségi információkat, végrehajtható parancsokat és megjegyzéseket tartalmazhat. A program mindent megjegyzésnek tekint, ami a # (andráskereszt) karaktertől a sor végéig tart. A megjegyzést nem tartalmazó sorok folytathatók a következő sorban a shell-eknél megismert sorvégi \ jellel.

A makródefiníciók azok a sorok, amelyek egy = jelet tartalmaznak, és a sort nem előzi meg sem tabulátor, sem kettőspont. A definíció formája a következő:

```
makró_név = string
```

ahol a makrónév tetszőleges betűvel kezdődő alfanumerikus karaktersorozat lehet. A string bevezető szóközei, tabulátorai figyelmen kívül maradnak. Ha a string elmarad, akkor a makró üres string lesz. A nem definiált makrók is üres stringnek értelmezőnek a makró kifejtésekor.

A makefile további sorai általános formában a következők:

```
cél1 [cél2] :[:] [feltétel1...] [;parancsok] [#...]
[<TAB>parancsok][#...]
```

ahol a cél1, cél2, feltétel1 tetszőleges karaktersorozatok, amelyek betűkből, számjegyekből, pontokból és / jelekből állhatnak. Ezekben lehetnek a shell által értelmezett állományhelyettesítő karakterek is (?, *). A parancsok tetszőleges karaktereket tartalmazhatnak (a # kivételével).

Egy cél több függőségi szabályban is szerepelhet, de ekkor csak két kettősponttal, ami lehetővé teszi, hogy alternatív függőséget fogalmazzunk meg. Ilyenkor csak azt a generálási szabályt fogja végrehajtani a program, amelyiknél először találta öregebbnek a célt valamelyik feltételnél.

A feltételekben szereplő állományok természetesen más szabályokban célként szerepelhetnek, így kialakítható egy függőségi fa, amit a make program be fog járni az elsődleges cél előállítása érdekében. Ha a függőségi fa bajárasa közben olyan ponthoz ér el a make program, amleyhez nem adtunk meg szabályt, úgy az ún. implicit szabályok valamalyikével probálkozik.

Az implicit szabályok nagy része előre adott, de mi magunk is írhatunk ilyet elő. Az implicit szabályok közül a legfontosabbak az ún. suffix (ragozási) szabályok. A make programnak megadható egy suffix lista, amelyet megpróbál eltávolítani az egyes cél nevekből. Az így kapott suffix alapján keres a suffix szabályban. Ilyen suffix lehet például egy file kiterjesztése. A suffix szabályok általános alakja:

```
FsCs:
<TAB>parancsok
```

Ahol **Fs** a feltétel suffix, **Cs** pedig a cél suffix. Így például egy **.c** kiterjesztésű forrás állományból a megfelelő **.o** kiterjesztésű object állomány előállításához szükséges parancsokat a következő implicit szabállyal adhatnánk meg:

```
.c.o:
        cc -c $<
```

A fenti szabályban a $< egy ún. dinamikus makrót jelöl. A makrók felhasználására a $(makró_név) ad lehetőséget. A () jel elhagyható, ha a makró neve egy karakterből áll.

A dinamikus makrók a jelentése a következő:

$*	A cél file nevének suffix nélküli része.
$<	A feltétel file neve.
$@	A cél file neve
$?	Azok a feltételek, amelyek újabbak a céltól.

A make program a szabályok leírását a **makefile**, **Makefile** vagy **MAKEFILE** állományban keresi, vagy explicit megadható indítási paraméterként a file neve (make -f file). A program a szabályokat leíró file feldolgozás után a legelső szabályban megadott cél objektumot, vagy az indítási paraméterként kapott célt probálja előállítani (make cél).

Példák:

1. Legyen a következő makefile:

```
prog:     x.o y.o z.o
     cc x.o y.o z.o -o prog
x.o: x.c x.h
     cc -c x.c
y.o: y.c x.h
     cc -c y.c
z.o: z.c
     cc -c z.c
```

   Ami a prog nevű programot állítja elő, ha bármelyik komponense megváltozott.

2. Makrók és implicit szabályok felhasználásával ezt így is írhattuk volna:

```
.SUFFIXES .c .o      #nem kell, mert deafult
OBJECTS = x.o y.o z.o
HEADS = x.h
prog: $(OBJECTS)
        cc $(OBJECTS) -o prog
$(OBJETS): $(HEADS)
.c.o:                #nem kell, mert default
        cc -c $<     #nem kell, mert default
```

7.2 SCCS, RCS

Az SCCS (Source Code Control System) lehetővé teszi, hogy egyetlen állományban tároljuk egy program forrásának több változatát. Az SCCS az

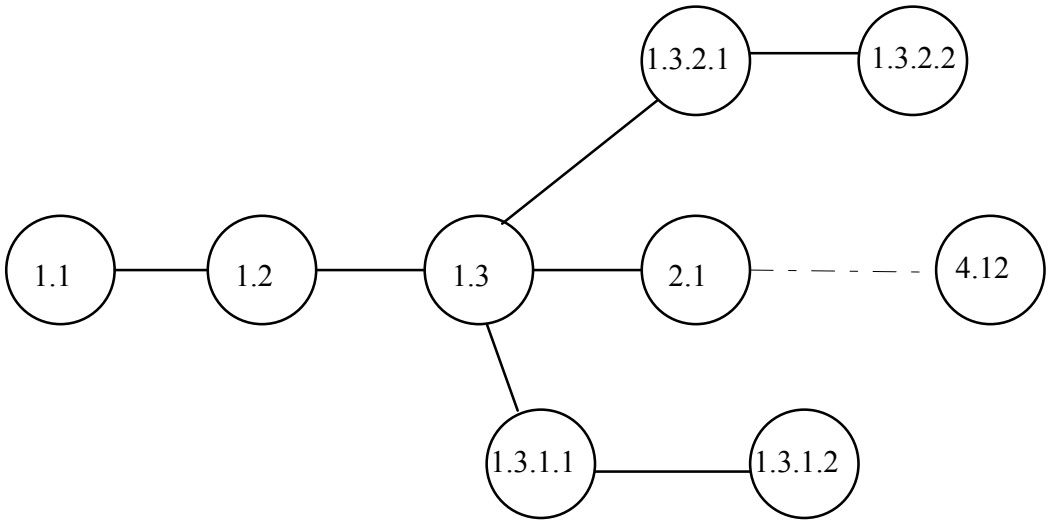
állományon történt változtatásokat, mint egy időben zajló sorozatot tárol el. Az állomány egyes változatait ún. deltának nevezi, ami nem pontosan, de majdnem megegyezik a verzió fogalmával. A deltákat egy számmal a SID-del (SCCS ID) jellemzi. Ez a szám maximum négy részből tevődik össze, amelyeket pont választ el egymástól:

R.L.b.l

ahol:

R	relase
L	level
b	branch
l	branch level

A legtöbb esetben a b és l számok elmaradnak, mert nincs branch. A branch szolgál arra, hogy egy egyébként egyenesen fejlődő programba elágazásokat tegyünk. Ilyenre szükség lehet, ha pl. a programot kiadtuk az X felhasználónak, aki felfedezett egy hibát, de közben a program már továbbfejlődött, és már egy új release-nél tartunk, ami már el is jutott az Y felhasználókhoz. Ha X valamilyen okból ragaszkodik ahhoz a relase-hez, és nem kell neki az új, amiben esetleg már nincs is ez a hiba, akkor kénytelenek vagyunk X release-ben kijavítani a hibát. Ekkor keletkezik a branch. Ezt a programfejlődési folyamatot szemlélteti a 7.1 ábra.



A 7.1 ábra egy program fejlődése

Az ábrán a program SID-je azt tükrözi, hogy két elágazás történt. Az első az 1.3.1.1, ami az 1. elágazás 1. változatát jelenti, és egy kicsit később az 1.3.2.1, ami ugyanabból az 1.3-as SID-ból származik, de ez már a második elágazás.

Az SCCS rendszer segítségével ezen változatok egyetlen egy állományban tárolhatók, amiből bármikor előállítható bármely változat. A rendszer továbbá biztosítja, hogy az egyes változatokhoz megjegyzéseket fűzzünk, ami később kiíratható. A mindig pontos adminisztráció érdekében lehetőség van néhány ún. SCCS változó használatára is, ami mindig aktualizálódik a forrásprogramban, ha

egy változatot előveszünk. Ilyen lehet pl. az aktuális dátum, a módosítás dátuma a SID stb.

A rendszer használata három parancs megtanulásából áll. Ez az **admin,** a **get** és a **delta.**

- Az admin az első sscs állomány előállítására és más adminisztrációs tevékenység ellátására szolgál.
- A get az sccs állományból előállítja a kívánt változatot. Normál esetben ez írásvédett, és csak akkor írható, ha a get parancsnak megadtuk a -e kapcsolót, ami az új delta előkészítését jelenti.
- A delta paranccsal lehetséges egy módosított változat behelyezése az sccs állományba.

A munka menete tehát a következő. Az admin paranccsal elkészítjük a legelső változatot az állományról vagy akár egy egész katalógus tartalmáról. Ha később csupán újra akarjuk fordítani a programot, akkor a get paranccsal elővesszük a legutolsó változatot. (Az admin letörli az eredetit, ezért az már nincs meg.) Ha módosítani is szeretnénk, akkor a get -e -vel előkészítjük az állományt javítani. Ezután javíthatunk. Ha előállt a megfelelő állapot, akkor a delta paranccsal becsomagoljuk azt az sccs állományba. A delta az admin-hoz hasonlóan letörli az állományunkat. Ezután a get-tel bármelyik korábbi változatot elővehetjük.

A rendszer használatát az sccs nevű front end program könnyíti, valamint az, hogy a parancsoknak komplett katalógusok is megadhatók, így nem kell egyenként minden állományt külön kezelni. Szintén a használatot könnyíti, hogy a make program beépített szabályokat tartalmaz az sccs állományokra.

Az **RCS** (Revision Control System) az SCCS-hez hasonló filozófiát követő rendszer. Lényegében az SCCS publikus változata, bár adatformátuma, parancsai, nem egyeznek meg az SCCS rendszerével, ezért azzal nem kompatibilis. Az RCS azonban a csoportmunkát is támogatja. Parancsai az **rcs, co, ci**:

- A **rcs** az első rcs állomány előállítására és más adminisztrációs tevékenység ellátására szolgál.
- A **co** (check out) az rcs állományból előállítja a kívánt változatot. Normál esetben ez írásvédett, és csak akkor írható, ha a ci parancsnak megadtuk a -l (lock) kapcsolót, ami az új delta előkészítését (zárolását) jelenti.
- A **ci** (check in) paranccsal lehetséges egy módosított változat behelyezése az rcs állományba.

Mindkét rendszer hasznos támogatást nyújt az egyes változatok azonosításához, a változatok történeti fejlődésének nyomon követéséhez. Ezt leginkább azzal éri el, hogy rákényszeríti a felhasználót, hogy néhány sorban adja meg az új változat létrejöttének okát is. Ez a későbbiekben bármikor lekérdezhető, kérésre a forrásba megjegyzésként automatikusan begenerálható.

7.3 Futási idő analízis

A UNIX rendszer operációs szinten támogatja a futási idő analízist. Egy igen egyszerű módszerrel a rendszer meg tudja mérni, hogy egy adott program mely

utasítások végrehajtásával mennyi időt töltött el. Ezt a különböző nyelvek fordítói ki tudják még olyan kódrészekkel is egészíteni, amely segítségével megszámlálható, hogy az egyes eljárásokat, függvényeket hányszor hívott meg a program. Ezen információk a program adatterületén gyűlnek, és a futás végén egy járulékos programrész kiírja egy állományba. Ennek alapértelmezés szerinti neve mon.out. A **prof(1)** program ezen állomány és a program szimbólumtáblájának ismeretében könnyen megérthető táblázatos statisztikát készít. Ezen statisztika optimalizálás szempontjából legfontosabb része a százalékos eloszlás. Ahol a program a futási idejének legnagyobb részét töltötte, ott kell megvizsgálni, hogy lehet-e valamilyen módszerrel gyorsítani a kódon.

Példa:

A következő példában egy C programot a fordító **-p** kapcsolójával fordítottuk le. Ennek hatására a programba a fordító automatikusan olyan kódrészleteket épített be, amelyek egyrészt bekapcsolták a UNIX futási idő analízisét, másrészt a rutinok hívási gyakoriságát is regiszrálták. A keletkezett program futtatásakor egy **mon.out** file keletkezik, amelyet a **prof** programmal feldolgoztunk, és a következő eredményt kaptuk:

```
 %time  cumsecs  #call  ms/call  name
 %time  cumsecs  #call  ms/call  name
  61.5     0.08      1    80.00  _vege
  38.5     0.13    500     0.10  _xyz
   0.0     0.13      1     0.00  _exit
   0.0     0.13      1     0.00  _main
   0.0     0.13      1     0.00  _printf
   0.0     0.13      1     0.00  _write
```

Az eredményben látható, hogy a program összes futási idejének 61.5%-át az _vege függvényben tölti, ami nem meglepő a forrásprogram ismeretében:

```
main(argc, argv)
int argc;
char *argv[];
{
   printf("Hello \n"); vege();
}

int vege()
{
   int i, j;
   for (j = 0; j < 500; j++) {
     for (i = 1; i < 1000; i++);
     xyz(j);
   }
   exit(0);
}

int xyz(i)
{
   float f1 = 3.14;
   while (i--) f1 += f1 * i;;
```

}

7.4 Kapcsolódó parancsok

A fontosabb programfejlesztő és karbantartó parancsok: (A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai.)

ar(1) - archivum (library) kezelő
as(1) - assembler
cc(1),gcc(1) - C fordító
dbx,adb,gdb(1)- debugger
ld(1) - tárgykód szerkesztő
make(1) - összefüggő programok és állományok konzisztens előállítása
m4(1) - makroprocessor
prof(1) - futási analízis készítése
rcs,ci,co(1) - RCS forrásprogram verzió követő
sccs,admin,delta,get(1)
- SCCS forrásprogram verzió követő

8. Unix hálózat

A UNIX operációs rendszerek hálózati működéséhez szinte kizárólag a TCP/IP hálózati technológiát alkalmazzák, melyet a 70-es évek közepén, a 80-as évek elején fejlesztettek ki az Egyesült Államokban a DARPA (Defense Advanced Research Project Agency) koordinálásával. A fejlesztési munkában a Californiai Berkeley Egyetemnek volt a legnagyobb szerepe. A kezdetben katonai célokat megvalósító rendszer a UNIX elválaszthatatlan részévé vált. A TCP/IP nem más mint, egy protokollcsomag[23], amely lehetővé teszi azt, hogy az alkalmazott fizikai átviteli közegtől függetlenül, akár kis-, akár nagytávolságú összeköttetés esetén a felhasználó ill. az alkalmazói programok egy egységes, jól definiált felületen keresztül kommunikálhatnak a partnerükkel. A TCP/IP protokollra épül az Internet technológia, amely nem más, mint az egymástól független lokális hálózatok egyetlen óriási, világméretű hálózattá való összekapcsolása, melyben a szomszéd szobában elhelyezett vagy a földrajzilag nagy távolságra lévő másik számítógép elérése azonos szabályok alapján történik. A TCP/IP és UNIX kettős az elmúlt 10-15 évben óriási karriert futott be az egyetemeken, az oktatásban, kutatásban és mérnöki munkahelyeken. Az Internet a világ nagyobb és leggyorsabban növekvő számítógépes hálózatává vált: kb. 2.5 millió számítógép van bekapcsolva az Internetbe, kb. 30 millió felhasználója van az összes kontinensen ill. több mint 100 országban.

A TCP/IP népszerűségét növeli, hogy mára már nemcsak a UNIX rendszerek támogatják, hanem az összes többi elterjedt operációs rendszerben (DOS, NOVELL, VMS, IBM OS, MAC OS, AMIGA DOS, stb.) is megvalósították, és így a heterogén rendszerek közötti kapcsolat alapjául is szolgál.

[23] Protokoll - azon szabályok gyűjteménye amelyek alapján a hálózatba kapcsolt gépek kommunikálnak egymással.

8.1 Címzés az Interneten

A TCP/IP rendszerben minden a hálózatra kapcsolódó egységnek saját Internet címe van. Az Internet címek egyediek, tehát elméletileg nincs a világon két egyforma Internet címmel rendelkező gép. (Ennek éppen a hálózat világméretűsége miatt van jelentősége!) Az Internet cím egy 32 bites szám, melyet a könnyebb kezelhetőség érdekében négy 8 bites tag együttesének formájában írunk le. Pl.:

152.66.76.1 vagy
152.66.77.2 vagy
143.108.1.1 stb.

A cím egyértelműen kijelöl a világon létező lokális hálózatok közül egyet, és azon belül egy adott gépet. Az Internet cím formája és tartalma független az adott lokális hálózatban alkalmazott fizikai technológiától (pl. ETHERNET, Token Ring, stb.), és nem tévesztendő össze a fizikai címmel! Különböző osztályú (A, B, C) Internet címek léteznek attól függően, hogy a 32 bitből hány bit jelöli ki a hálózatot, és hány a hálózaton belüli gépet. Az Internet címekhez a jobb megjegyezhetőség érdekében neveket is rendelnek. Egy címet tehát nemcsak számokkal, hanem nevekkel is megadhatunk. Ez a domain-név címzés. A domain név ponttal elválasztott több (>= 2) tagból áll, pl.:

fix.fsz.bme.hu vagy
sun10.vsz.bme.hu vagy
fpsp.fapesp.br stb.

A domain-név a szervezeti egység szerint hierarchikus, és a hierarchiaszintek száma nincs megkötve. Ha egy konkrét gépet akarunk megcímezni, akkor az első tag a gép neve. Egyébként a szokásos felépítés:

gép.osztály.intézmény.ország

A domain-név egyes tagjai és a számmal adott Internet cím egyes tagjai között NINCS egyértelmű megfeleltetés! A legtöbb alkalmazás megengedi, hogy a felhasználó a számmal megadott cím helyett a megfelelő nevekkel leírt címmel hivatkozzon az elérni kívánt gépre. A domain-név -> Internet cím leképezést az ún. domain-név szerverek rendszere valósítja meg.

8.2 Kliens - szerver modell

A kliens - szerver vagy ügyfél - szolgáltató modell szinte a UNIX minden szolgáltatásában megtalálható, de a legszembetűnőbb a hálózati szolgáltatásoknál. A kliens mindig egy adott szolgáltatás igénylőjét a szerver pedig ennek a szolgáltatóját jelenti. A szervereket többnyire ún. daemon programok valósítják meg, melyek várakoznak az egyes kliensek jelentkezésére ill. kéréseire, majd végrehatják azt. Ilyen daemon pl. a nyomtatást végző line printer daemon, vagy a távoli terminálszolgáltatást nyújtó telnet daemon. Természetesen a hálózati szolgáltatásokat megvalósító szerverek és a szolgáltatást igénybe vevő kliensek valamilyen jól definiált protokollal a hálózaton keresztül kommunikálnak

egymással. A szerver programok rendszerint a UNIX rendszer részét képezik és indításuk automatikus. A kliens programot pedig többnyire a szolgáltatást kérő felhasználó indítja.

8.3 Az Internet

Az Internet olyan világméretű számítógép-hálózat, amely az egyes tagországok egyes tagintézményeinek lokális hálózatait kapcsolja össze egyetlen nagy egységgé. Az Internet legfőbb központja Amerikában van, de minden tagországnak is van egy saját Internet központja (Internet backbone), amely az adott országban felelős a hálózati üzemért, a címek és felső szintű domain nevek kiosztásáért. A domain nevek utolsó tagja az ún. top level domain, amely elárulja, hogy mely országbeli címről van szó. (pl.: **hu** - Magyarország, **de** - Németország, **es** - Spanyolország, **br** - Brazília, stb.) Az Egyesült Államokbeli gépek címeinek utolsó tagja rendszerint nem országkód, hanem valami egyéb kategória. (**com** - kereskedelmi felhasználók, **edu** - oktatási intézmények, **gov** - kormányhivatalok, **mil** - hadsereg, **net** - hálózati központok, **org** - egyéb szervezetek)

8.4 Az Internet fontosabb szolgáltatásai

Az Internet rendkívül széleskörű, sokszínű szolgáltatásokat nyújt a felhasználóknak. A TCP/IP segítségével megvalósítható a számítógép-hálózatok négy legfontosabb alapfunkciója: elektronikus üzenetváltás (levelezés), file transzfer, távoli terminál funkció, erőforrások (file rendszerek) megosztása. A hagyományos hálózati alapfunkciókon kívül lehetőség van ún. levelezési listák, elektronikus újságok, adatbázisok, archívumok, multimédia alkalmazások elérésére, a felhasználók közötti on-line beszélgetésre, stb.

8.4.1 Levelezési szolgáltatás

Az egyik leggyakrabban használt szolgáltatása az Internetnek az elektronikus levelezés (e-mail). A legnagyobb vonzereje az e-mail-nek, hogy a világ másik pontjára rendkívül gyorsan, akár néhány perc alatt küldhetünk üzeneteket, amelyek szinte bármit tartalmazhatnak. Ez azt jelenti, hogy az elektronikus levelek nemcsak szöveget, hanem képeket, hangot és akár általunk definiált csatolt dokumentumokat is tartalmazhatnak. Az elektronikus levelezésben - a hagyományos postai levelezéshez hasonlóan - két fontos résztvevő van: a leveleket író és olvasó felhasználó és a levelek továbbításáért ill. kézbesítéséért felelős szolgáltató (posta). A levelek írását és olvasását különböző fajta ún. front-end programok segítik: **mail, elm, pine, mailtool, DOS környezetben pmail,** stb. Ezek a programok eltérő fejlettségű kényelmi funkciókkal (beépített szövegszerkesztő, címlista, címtár stb.) segítik a felhasználót. A leveleket továbbító programok (postások) hagyományosan az SMTP (Simple Mail Transfer Protocol) protokollt használják. Sajnos az SMTP csak a 7 bites adatátvitelt támogatja, ezért az olyan adatokat, amelyek a 8. biten is hordoznak hasznos információt valamilyen eljárással 7 bites adatokra kell bontani és a rendeltetési helyen pedig visszaalakítani. Ezt a célt szolgálja az igen elterjedten használt **uuencode - uudecode** programpáros, amelyet egyes levelező front-end-ek beépítetten tartalmaznak. Másik megoldása lehet a problémának az

ún. MIME (Multipurpose Internet Mail Extension) ajánlás szerinti kódolás, amely nem csak a kódolásra ad módszert, hanem a megjelenítéshez is nyújt információt. Megjegyezzük, hogy az SMTP kiterjesztett változata az ESMTP (Extended Simple Mail Transfer Protocol) már támogatja a 8 bites átvitelt, de egyenlőre az SMTP használata van elterjedve az Internet hálózaton.

Az elektronikus levélen keresztül el lehet érni egyéb szolgáltatásokat is. Az egyik ilyen szolgáltatás például a **listaszerver**. A listaszerverek a címükre küldött leveleket mindenkinek elküldik, akik az adott szerverre "feliratkoztak". A feliratkozás egy megfelelő címre elküldött levéllel történik. Így az azonos érdeklődési körű emberek egy-egy ilyen listára feliratkozva egyszerűen megbeszélhetik problémaikat. Lényegében konferencia jellegűen érintkezhetnek egymással. Ilyen listák a legkülönfélébb témakörökben léteznek, és bárki egyszerű eszközökkel újakat hozhat létre. Egy másik, szintén levelezéssel elérhető szolgáltatás a különböző **program- és adatarchívumokhoz való hozzáférés**. Ezen szolgáltatás lényege, hogy megfelelő formátumú levelet küldve egy adott címre adatokat kérhetünk az archívumból, amelyeket szintén levélben kapunk meg. Természetesen a levélben elküldött kérésünket egy program dolgozza fel, és válaszol. Így a lekérdezett adat néhány perc alatt már meg is érkezhet a válaszlevélben. Ehhez hasonló szolgáltatás az általánosan FTP-vel elérhető archívumok lekérdezésére tervezett ún. ftpmail szerver szolgáltatás.

8.4.2 File transzfer szolgáltatás

Az Internet másik nagyon fontos szolgáltatása az FTP (File Transfer Protocol). Ezen szolgáltatás segítségével egyszerűen másolhatunk állományokat a helyi gépünkre távoli gépekről, vagy fordítva. A szolgáltatást valójában két program együttműködéseként érhetjük el. A szerver vagy szolgáltató gépen működő FTP daemon a hozzáférési jogosultság ellenőrzése után kiszolgálja a kliens vagy ügyfél kéréseit. A kliens program standard neve **ftp**. Normál esetben a távoli gépen is kell, hogy legyen témaszámunk. Az FTP-nek azonban van egy nyilvános szolgáltatása is, az anonymous FTP. Ennek igénybevételéhez anonymous néven kell bejelentkezni az adott szerverre és a password kérdésre a saját e-mail címünket illik megadni. Az anonymous FTP szervereken az állományok tartalom szerint katalógusokba vannak rendezve, akár mint egy Unix-os file-rendszerben, vagy akár mint egy BBS-en. Fontos különbségek BBS-ekkel szemben, hogy általában sokkal rendszerezettebben vannak ezeken a szervereken az adatok, és hogy az archív szerveren csak legális, tehát shareware, freeware vagy public domain programokat és adatokat találunk. Erre kötelezi ezen szerverek üzemeletetőit a Netiquette is. Attól azonban, hogy így nincs semmi információ ezeken a szervereken nem kell tartani: nem ritkák a több Gigabájt méretű (akár 100 Gbyte!) méretű archívumok.

8.4.3 Távoli bejelentkezés

Távoli terminálszolgáltatást a Telnet vagy Rlogin protokoll segítségével vehetünk igénybe. Ezzel a szolgáltatással a távoli számítógépre a hálózaton keresztül jelentkezhetünk be. A hálózat “átlátszó”, azaz ugyanúgy dolgozhatunk mint a saját gépünkön dolgoznánk. Ilyen szolgáltatással nyilvános témaszámokon keresztül

közérdekű szolgáltatásokhoz, könyvtári rendszerekhez, adatbázisokhoz férhetünk hozzá. Ilyen lekérdezési módot alkalmaznak több IIF adatbázis elérésére is. Fontos tudni, hogy a **telnet** használatakor mindig megtörténik a jelszavas hozzáféres ellenőrzés, de az **rlogin** esetében a távoli témaszám tulajdonosa dönthet úgy is, hogy bizonyos felhasználóknak (pl. saját magának) megadott gépekről megengedi a jelszó nélküli hozzáférést. Ezt a login katalógusban elhelyezett **.rhosts** nevű állomány megfelelő kitöltésével lehet elérni. Pl a

```
fix.fsz.bme.hu          jozsi
```

sor engedélyezi, hogy a fix.fsz.bme.hu nevű gépről bejelentkező jozsi nevű felhasználótól az rlogin program ne kérdezzen jelszót. Ez azért fontos, mert néhány távoli végrehajtást végző program is ezt az állomány használja:

```
rcp     -       távoli és a helyi gép közötti file másolás
rsh     -       távoli gépen történő parancsvégrehajtás
                (egyes rendszereknél remsh, ahol az rsh a restricted sh)
```

Biztonsági okokból a .rhosts védelmi attribútumának 600-nak kell lennie, amit egyes rendszerek szigorúan ellenőriznek.

A biztonsági okokból azonban sem a telnet, sem az rlogin protokoll önmagában nem nyújt megfelelő védelmet az esetleges hálózati forgalom lehallgatása ellen. Ezért ahol erre lehetőség van, és nem üzemel más biztonsági rendszer, célszerű az **ssh** programot használni a távoli bejelentkezésre és a távoli parancsvégrehajtásra. Az ssh nyílt kulcsú kódolást alkalmaz, ami megfelelő beállításokkal kellő védelmet nyújthat jelszavaink lehallgatása ellen.

8.4.4 Hírlevél szolgáltatás

A levelezés után a légszelesebb körben használt Internet szolgáltatás a Usenet News. A Usenet News több mint 4000 újság, témakörök szerint csoportosítva. Téma választéka hasonlít a levelezési listákhoz, annál is inkább mert a levelezési listák egy része bekerül ezekbe az újságokba. Nagy különbség, hogy míg a levelezési listáknál addig rajta vagyunk a listán míg nem iratkozunk le róla, itt vagy elolvassuk az újságot, vagy nem. Az újságokat az news szerverek őrzik. Ezek ún. együttműködő szerverek, melyek elvégzik az újságok frissítését, azaz az új cikkeket továbbítják egymásnak. Egyedül a news szerveren múlik, hogy mennyi ideig tartja meg a cikkeket. A fontosabb newsgroup-ok archívumba szoktak kerülni általában egy anonymous FTP szerveren. A leggyakrabban feltett kérdésekre (FAQ - Frequently Asked Questions) a megadott válaszokat szintén archívumokban tárolják. Az újságok (newsgroup-ok) hierarchikusan vannak szervezve például tudományos - **sci**, számítástechnikai - **comp**, szabadidő és hobbi - **rec**, közösség és életmód - **soc**, egyéb - **misc** és alternatív - **alt**. Ezeken a nemzetközi újságokon kívül létezhetnek regionális és nemzeti újságok is, így például a magyar nyelvű újságok a **hun** csoportnév alatt találhatók. A nemzeti csoportok száma felülmúlja nemzetköziekét. A leggyakrabban használt news olvasó programok: **tin, rn, trn, nn, xvnews, xrn**. Néhány WWW kliens program is lehetővé teszi a news olvasását.

8.4.5 Adatbázis és információs szolgáltatások

A **gopher** (pocok) program egy világméretű, az Interneten elosztott adatbázis elérését teszi lehetővé. Az adatbázis egyes részeit a gopher szerverek tartalmazzák. A szervereket megfelelő kliens programokkal, ún. Gopher protokoll segítségével lehet elérni és lekérdezni. A felhasználó egy menü rendszerű interfészt lát. A menü egyes pontjai újabb menükre, dokumentumokra vagy Telnet kapcsolatokra mutathatnak, amelyek esetleg a világ egy másik pontján helyezkednek el. A dokumentumok szövegeket, információkat, bináris állományokat tartalmazhatnak. A menüs kezelői felület eltakarja a felhasználó elől, hogy valójában honnan is érkezik a kért dokumentáció.

A WWW (World Wide Web - világméretű pókháló) az Interneten létrehozott, elosztott, ún. hipermédia adatbázis. A hipermédia kifejezés arra utal, hogy az adatbázis tartalmazhat formázott, strukturált (hypertext) szöveget, ábrákat, fekete-fehér és színes képeket, sőt akár hangzó és mozgóképi információkat is (multimédia). A lekérdezés az Internetbe kapcsolt számítógép bármelyikéről egy megfelelő kliens program segítségével lehetséges. A WWW az új, multimédia formában tárolt információkon túl integrálja a már korábban létrehozott Internet adatbázisokat és információs rendszereket is (pl. gopher, news, ftp vagy akár telnet). Ez azt jelenti, hogy egyetlen kliens program használatával, azonos felhasználói felületen keresztül bármelyik szolgáltatást elérhetjük. Ehhez csupán a megfelelő információ forrás helyét és az elérés protokollját kell ismerni. E két információból egy ún. URL (Universal Resource Locator) állítható össze, amit a kliens program megért és a megfelelő protokollal lekérdezi a kért információt. Néhány példa az URL-re:

http://www.fsz.bme.hu/hungary/index.html

Az elérési protokoll a WWW protokollja a *HTTP* (Hyper Text Transfer Protokoll), a dokumentum a *www.fsz.bme.hu* gépen helyezkedik el a *hungary* katalógusban, a neve pedig *index.html*.

ftp://ftp.fsz.bme.hu

Az elérési protokoll az *FTP*, a dokumentum pedig az *ftp.fsz.bme.hu* gépen levő fő dokumentum (katalógus).

gopher://gopher.bme.hu:70/11/hun/tel-adat

A elérési protokoll a *Gopher* protokoll, a dokumentum a *gopher.bme.hu* nevű gépen a *70*-es porton működő gopher szerver *11/hun* katalógusában levő *tel-adat* nevű állomány.

Jelenleg világszerte több mint 6000 WWW adatbázis szerver működik. Az adatbázisok tartalma elképesztően változatos. A legismertebb lekérdező programok: **netscape, mosaic, arena, lynx**.

8.4.6 Egyéb Internet szolgáltatások

Gyakran használt szolgáltatása az Internetnek a **talk** és az **irc**. Az előbbi egy telefon jellegű on-line szolgáltatást jelent. A különbség a telefon és talk között annyi, hogy gépelni kell azt amit mondani akarunk. Az IRC (Internet Relay Chat) leginkább a CB rádiózáshoz hasonlít: Be lehet kapcsolódni a beszélgetésekbe, át lehet lépni más csatornákra stb. Az IRC csatornákon gyakran szoktak konferenciákat rendezni különböző témákról.

Mivel az Interneten olyan sok információ áll rendelkezésre, hogy nehéz megtalálni a nekünk szükségeset, ezért fejlesztettek ki sokféle kereső szolgáltatást: pl. Archie, Veronica, Jughead, Wais, Hytelnet, Netfind, Whois, WebCrawler, Lycos, WWWW stb. Ezen szolgáltatásokat megvalósító szerverek vagy külön lekérdező programmal vagy gopher ill. WWW kliens programba beépített felülettel érhetők el.

Az **archie** programmal az anonymous szerverek tartalmára kereshetünk rá, ha tudjuk, hogy körülbelül milyen file-t keresünk. A Veronica és a Jughead szolgáltatások a "Gopherspace"-ben való keresgélésre szolgálnak. A **waissearch** programmal az Interneten elhelyezkedő nagyszámú szöveges adatbázisban kereshetünk. A Hytelnettel a gopher felületén keresztül több száz telnet szolgáltatás katalógusa (időjárásjelentés, hírek, információk, interaktív játékok, stb.) kérdezhető le. A **whois** program és a telnet kezelői felületen keresztül elérhető Netfind szolgáltatás e-mail, postai címek és telefonszámok keresésére szolgál. A WebCrawler, Lycos, WWWW segítségével a WWW szerverek adatbázisaiban kulcsszó szerint kereshetünk. A hatalmasra duzzadt WWW hálóban kategorizált indexeket is építenek az információ keresés megkönnyítésére. Ilyenek pl: Yahoo!, Jumpstation, NIKOS. Ezek egy része elérhető a következő dokumentumtól kiindulva:

http://www.fsz.bme.hu/internet/discover.html

8.5 Hálózati állományrendszer

A Sun Microsystems által a 80-as évek közepén kifejlesztett NFS (Network File System) segítségével lehetőség nyílik a TCP/IP hálózatba kötött gépek állományrendszerének összekapcsolására. A felhasználó számára a távoli gép állományrendszere - a lokális kötetekhez hasonlóan - a lokális gép fa-struktúrájának egy pontjára kapcsolódva jelenik meg. Így a távoli állományok elérése ill. kezelése a lokális állományokkal teljesen megegyező módon történik. A távoli és a lokális állományrendszer össze- ill. szétkapcsolása (mount) igény szerint automatikusan is történhet. A működést a UNIX rendszerrel együttműködő daemon programok biztosítják. A távoli állományok virtuális i-node-okkal (v-node) vannak azonosítva. Mivel a felhasználók azonosítása az i-node-okban tárolt uid és gid alapján történik, az NFS-be kapcsolt számítógépeknek közös, vagy szorosan összehangolt felhasználói adminisztrációjának kell lennie. Ezt segíti a NIS (Network Information Server), amely hálózati információs adatbázis kezelését teszi lehetővé. Az NFS rendszer biztonsági és teljesítmény korlátainak kiküszöbölésére más fajta hálózati állományrendszerek is születtek pl. AFS, DFS.

8.6 Kapcsolódó parancsok

A fontosabb hálózattal kapcsolatos parancsok: (A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai.)

archie(1) - keresés ftp archívumok tartalmában
finger(1) - felhasználó adatainak lekérdezése
ftp(1) - file transzfer gépek között
gopher(1) - gopher kliens
irc(1) - on-line konferencia hálózaton keresztül
mail,pine,elm(1) - elektronikus levél küldése/fogadása
mount(8) - kötetek bekapcsolása az állományrendszerbe
netscape,mosaic(1) - WWW böngésző
lynx(1) - alfanumerikus WWW böngésző
ping(1) - távoli gép elérhetőségének tesztelése
rcp(1) - file másolás a távoli és a helyi gép között
remsh(1) - távoli parancsvégrehajtás
rlogin(1) - másik gép elérése távoli terminálként
rsh(1) - távoli parancsvégrehajtás
rn, tin, trn(1) - news olvasó/író
ssh(1) - másik gép elérése távoli terminálként (kódolt adatátvitel)
ssh-keygen(1) - nyílt - titkos kulcspár generálása az ssh kapcsolatfelvételhez.
showmount(8) - kötetek mount paramétereinek kiírása
talk(1) - on-line beszélgetés a hálózaton keresztül
telnet(1) - másik gép elérése távoli terminálként
uudecode(1) - uuencode-dal kódolt állomány dekódolása
uuencode(1) - bináris file kódolása 7 bites formába
umount(8) - kötetek lekapcsolása az állományrendszerről
whois(1) - e-mail, postai címek és telefonszámok keresés

9. Grafikus felhasználói felület (X)

A számítástechnikai eszközök gyors fejlődése lehetővé tette, hogy a 70-es években még gyakran használt írógép jellegű teletype-okat hamarosan felváltották a katódsugárcsöves alfanumerikus, majd grafikus megjelenítők. A 80-as évekre már egyre több gyártó jelent meg olyan programmal a piacon, amely grafikus felhasználói felülettel rendelkezett. Ezek azonban az adott géptípus grafikus sajátosságait kihasználva készültek, és igen nagy munkaráfordítással lehetett csak más gépre átvinni azokat. Természetes igénnyé vált egy egységes grafikus rendszerfelület kifejlesztése. 1994-ben az MIT-ban (Massachusetts Institute of Technology) létrehoztak egy Athena elnevezésű projektet, melynek egy grafikus ablakozós felhasználói felület kifejlesztése volt a célja. A kezdetektől aktív rétvevője volt a projektnek a DEC (Digital Equipment Corporation) cég is. A fejlesztést a Stanford University W nevű ablakozós programjának felhasználásával kezdték és lényegében ebből alakult ki a mai **X Window[24] System**. Az első széleskörben hozzáférhető változat 1986-ban jelent meg X10.4 néven. A követező nagy lépés 1987-ben volt, amikor az X11.1 megjelent. Ez a kiadás sok új elemet tartalmazott, így pl. lehetővé tette több képernyő alkalmazását, bővíthetővé vált stb. Maga az X protokoll is megváltozott ami kompatibilitási gondokhoz vezetett. A 10-es főváltozat azonban hamarosan kihalt, a 11-es pedig dinamikusan fejlődött tovább. Kezdetben gyakran követték egymást az újabb változatok, így pl. 1988 elején már megjelent az X11.2, év végén pedig a 11.3. Azonban maga a 11-es főverzió stabilnak mutatkozik és egyre ritkábban változik. Ma az X11.6-nál tartunk, amit 1994-ben adtak ki (az X11.5-ot 1991-ben). Az egyes alváltozatokról a magasabb irányba való áttérést az X szubrutin könyvtár (Xlib) biztosítja. A rendszer nyitottságát mutatja, hogy az újabb alváltozatokba folyamatosan beépülnek a fontosabb előírások ill. szabványok. Így épült be pl. az X11.6-ba a Kerberos előírás.

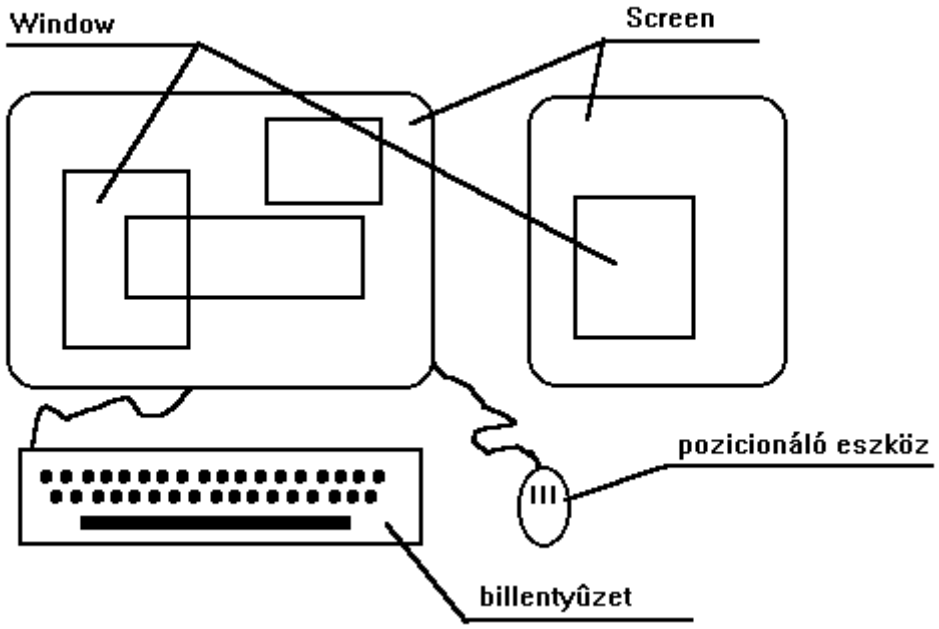
Az X rendszer elterjedésének, szabvánnyá válásának és a gyártók általi elfogadottságának egyik fontos tényezője, hogy 1988-ban számos nagy számítógépgyártó támogatásával létrehozták az X Konzorciumot, melynek legfontosabb feladata az X szabvány támogatása, terjesztése. A rendszer forráskódja szabadon hozzáférhető. Napjainkra elválaszthatatlan részévé vált a UNIX rendszernek, de léteznek nem UNIX alatti implementációk is mint pl. a VMS alatti DECWindow rendszer.

9.1 Az X rendszer felépítése, alapfogalmak

A rendszer egyik legfontosabb eleme a grafikus szolgáltatásokat nyújtó - egy vagy több grafikus képernyőt, billentyűzetet és pozícionáló eszközt (egér, tablet stb.) magába foglaló - munkahely, X terminológiával **display**. A rendszer működése az előző fejezetben már említett kliens - szerver modellen alapul: A grafikus munkahely szolgáltatásait egy a munkahely eszközeit vezérlő **szerver** programon keresztül vehetik igénybe a **kliens** programok. Egy kliens egyszerre több szerverrel is tarthatja a kapcsolatot ill. egy szerver több klienst is ki tud szolgálni egyidőben. A kliens és a szerver a hálózat bármely pontján lehet. Így a hálózat felett elosztott

[24] Sajnos több szakirodalom hibásan windows-t használ window helyett.

rendszerről beszélhetünk, mivel a számítás és a megjelenítés más gépen történhet. A kommunikáció egy ún. **X protokoll** felhasználásával történik, ami magasszintű protokoll és egy megbízható adatátvitelt biztosító alacsonyabb szintű protokoll (pl. TCP/IP, DECnet) meglétét feltételezi. Egy (X) display-hez több grafikus megjelenítő (képernyő) is tartozhat. Ezeket **screen**-eknek nevezzük. Több screen esetén is csak egyetlen billentyűzet és egy pozícionáló eszköz (pl. egér) tartozik a display-hez, ami az egyik képernyőről a másikra áthúzható.



9.1. ábra. Az X display

A kliens programok adatai különböző ablakokban - **window** - jelennek meg. Természetesen egy klienshez több window is tartozhat. Az ablakok az X szerverben mint erőforrások - **resource** - jelennek meg. Hasonló szerver erőforrás pl. a font, cursor, grafikus környezet stb. A szerver erőforrások a hálózati forgalom csökkentésének egyik alapeszközeként szolgálnak, ugyanis a kliens az erőforrások rövid azonosítójával képes hivatkozni a szerverben megvalósított bonyolult adatstruktúrákra. Az ablak más szempontból is érdekes fogalma az X rendszernek. Az ablakok ugyanis egy a UNIX állományrendszeréhez hasonló hierarchikus fába rendezhetők, mégpedig egy családfába. Így a főablakot (**root windows)** kivéve, minden ablaknak van szülő ablakja, és csak a szülő területén belül levő része látszik. Vannak ablakok, melyek csak bevitel céljaira szolgálnak, vannak melyek ki- és bevitelre egyaránt alkalmasak. Egy ablakban bevitel csak akkor történhet, ha nála van a bevitel vezérlésének joga, azaz az **input focus**.

9.2 Ablakkezelés

Fontos filozófiája az X rendszernek, hogy a kliensek nem vezérlik közvetlenül az ablakok méretét ill. pozícióját. Az ablakok elhelyezkedését, méretét, átfedését egy speciális kliens program, a **window manager** vezérli. Érdekessége az X

rendszernek, hogy valójában a window manager program is éppen olyan kliens, mint a többi kliens program (picit több joggal a szerver felé), és bármikor lecserélhető másikra[25], sőt nem is feltétlenül kell léteznie. Az X rendszer tervezői még az ablakkezelési stratégiát sem határozták meg, így például azt sem, hogy egyáltalán átfedhetik-e egymást az ablakok vagy sem, vagy pl. mikor és hogyan kerülnek át az input események (input focus) az egyik ablakból a másikba. Ennek több oka volt:

- Úgy gondolták, hogy nincs kialakult ablakkezelési stratégia, és majd a tapasztalatok döntenek, hogy mi a megfelelő.
- Az ablakok kezelési stratégiája, megjelenése nagymértékben meghatározza az egész rendszer külső megjelenését, azaz a felhasználói felületet, amit leginkább befolyásolni akartak ill. akarnak az egyes gyártók. Ha ez szabadon kialakítható, úgy a korábban kialakult gyártói stílushoz, ill. hagyományokhoz illeszkedő felület egyszerűen átvihető az X rendszer alá. Ez nagyban segítette az X rendszer elfogadását ill. elterjedését.
- A felhasználói felület ezáltal még inkább nyitott lett.

Ennek a nyitottságnak természetesen az lett az ára, hogy több különböző ablakkezelő program keletkezett és terjedt el. Ezek közül a legjelentősebb a Motif Window Manager és az Open Look Window Manager. Az előbbit a számos gyártót tömörítő OSF, míg az utóbbit az AT&T és a Sun támogatja ill. csak támogatta, mivel a Sun Solaris rendszere 1993-tól már a Motif irányzatot képviseli. Ezzel nagyrészt eldőlni látszott a két nagy rivális rendszer harca. Az igazi nagy lépés az egységesítés felé azonban a **CDE** megjelenése volt. A CDE (Common Desktop Environment) az OSF által támogatott grafikus környezeti szabvány, amely alapvetően a MOTIF rendszer filozófiáját követi, de egyes OpenLook tulajdonságokat is magában hordoz. Jelenleg ez a UNIX rendszerek gyártói által széleskörben elfogadott és támogatott grafikus rendszer. Elterjedtségük miatt azonban a MOTIF és az OpenLook rendszerek még sokáig fenn fognak maradni. A következőkben felsorolunk néhány elerjedtebb ill. érdekes ablakkezelőt:

uwm	Universal Window Manager, amelyet az MIT a kezdeti rendszerekhez adott.
twm	Tab Window Manager. X11.4-től ez a hivatalos ablakkezelő az MIT csomagban. Ez valamivel szebb és kényelmesebb mint az uwm..
rtl	A Siemens érdekes, de nem elterjedt ablakkezelője, amely nem engedte meg az ablakok tartós átlapolódását.
mwm	Motif Window Manager
olwm	Open Look Window Manager
fvwm	Virtual Window Manager. A twm továbbfejlesztett változata, amely virtuális screen-ek kezeléset is lehetővé teszi.
dtwm	CDE Window Manager

[25] Akárcsak a shell-ek esetében elvileg bárki készíthet magának saját ablak menedzsert.

9.3 Eseményvezérlés

Az X rendszer eseményvezérelt elven működik, amely elsősorban a rendszer programozása szempontjából lényeges. Ez azt jelenti, hogy a kliens program különböző eseményekre reagál, és nem a hagyományos módon vezérli a megjelenítést. Eseményként jelenik meg a felhasználói felületről érkező beavatkozás pl. billentyű nyomás, de eseményt küldhet egy másik kliens is pl. a window manager. Ilyen esemény érkezik például, ha megszűnik a kliens valamelyik ablakának az átfedése és az részben, vagy egészben újból láthatóvá válik. Ekkor a kliens program köteles a láthatóvá vált részletet újrarajzolni, és ezzel a képernyő megfelelő részén megjelenik a várt ábra. Hasonlóan az ablakkezelőtől jövő üzenet jelzi, hogy ha egy ablakot a felhasználó kicsinyíteni, vagy nagyítani kívánja. Az eseményvezérlés réven valóban a felhasználó vezérli a programot és nem fordítva.

Érdekessége a rendszernek, hogy minden ablakokhoz egy ún. esemény maszk tartozik, amely előírja, hogy mely eseményekre reagáljon az adott ablak. A nem kezelt események az ablakok hierarchiáján a szülő ablak felé terjedve újabb és újabb ablakokhoz juthatnak, vagy kérésre eldobódnak.

9.4 Az X rendszer indítása, használata

Az X rendszer kezelői felülete, használata nagyon sokban hasonlít más ablakozós rendszer felületéhez (pl. MS Windows), ugyanakkor sokban el is tér attól. Jelen segédlet keretei nem adnak módot a kezelői felület részletes ismertetésére, így csupán néhány érdekesebb dolgot ragadunk ki. Elsőként nézzük a rendszer indítását:

Általában a UNIX rendszerű munkaállomásokra való bejelentkezéskor azonnal elindul valamilyen X felület. Sokszor maga a felhasználó azonosítása (név, jelszó) már grafikus felületen történik. Vannak azonban esetek, amikor a grafikus felületet külön parancsokkal kell elindítani. A probléma két fázisra bomlik: Elsőként valahogyan el kell indítani az X szervert. A második fázisban pedig a kliens programokat kell elindítani, amelyekkel közölni kell, hogy melyik szervert használják. A következő esetek lehetségesek:

- Ha a szerver program még nem fut az adott UNIX rendszeren, akkor azt az **xinit** paranccsal lehetséges elindítani. Ekkor természetesen a felhasználó azonosítása még az alfanumerikus felületen történik, és azután adható ki az xinit parancs. A szerver indítása után végrehajtódik a felhasználó login katalógusában a **.xinitrc** shell script, amely az elsőként elindítandó kliens programot(kat) hivatott elindítani. Rendszerint ez a shell script indítja a window manager-t is, mégpedig úgy, hogy azt előtérben indítja, míg a többi klienst háttérben. Ha .xinitrc állomány nincs, akkor automatikusan az **xterm** program indul első kliensként.
- Ha X terminál emulációt, vagy X terminált használunk akkor a szerver a berendezés bekapcsolásakor, vagy a programcsomag elindításakor elindul. Ekkor fel kell venni a kapcsolatot valamilyen módon a gazda-számítógéppel, amelyen a kliens programjainkat kívánjuk majd futtatni. Ez a kapcsolatfelvétel rendszerint XDMCP-vel (X Display Manager Control

Protokol) történik. Ekkor a gazda-számítógépen futnia kell az **xdm** programnak, amely konfigurációtól függően felkínálja az XDMCP-vel elérhető számítógépek listáját (chooser), majd a felhasználó azonosítása után végrehajtja a kiválasztott gépen a konfigurációban meghatározott shell script-et (pl: /usr/lib/X11/xdm/Xsession). Ez alapesetben az adott felhasználó login katalógusában levő **.xsession** nevű állományt hajtja végre[26] Előfordulhat azonban, hogy a gazda-számítógépen nem fut xdm. Ekkor telnet, rlogin, vagy rsh protokollal történhet az első kliens program indítása, ami általában szintén xterm.

- A legtöbb grafikus munkaállomáson az X szerver a munkaállomás indításakor automatikusan elindul. Ezekben az esetekben a felhasználó azonosítása szintén az xdm segítségével történik.

Mind az xinit, mind az xdm a végrehajtott script futásának végéig vár. Ha ez véget ér, akkor az xinit megállítja az xdm pedig reszeteli az X szervert. Ez a magyarázata annak, hogy ezek a script-ek a legutoljára indított klienst - ami rendszerint a window manager - előtérben indítják. Ha ez a program megáll, a script is megáll. Ekkor az összes, a szerverhez kapcsolódó kliens futása befejeződik. A következőkben bemutatunk egy-egy példát a .xinitrc és a .xsession állományra:

```
# minta .xinitrc
xclock -geometry 50x50-0+0 -bw 0 &
xterm -geometry 80x24+0+0 &
xterm -geometry 80x24+0-0 &
mwm
```

```
# minta .xsession
twm &
xclock -geometry 50x50-0+0 -bw 0 &
xterm -geometry 80x4+0+0
```

Az első példában a .xinitrc végén a window manager indul előtérben. Ennek az lesz a hatása, hogy ha az mwm megáll, akkor a .xinitrc is meg fog állni. A második példában a .xsession végrehajtása akkor fog megállni, ha az xterm program megáll.

Az **xterm** program egy alfanumerikus terminál szolgáltatásait nyújtja némi grafikus kiegészítésekkel (scroll bar, menü, stb). Ezen felületen az adott felhasználóhoz rendelt parancsértelmező (shell) jelentkezik, ami fogadja és végrehajtja a parancsokat.

A szerver indítása után következhet a különböző kliensek elindítása, melyek bizonyos indítási paramétereket egységes formában kezelnek. Ezt részben az Xlib biztosítja, részben a megfelelő programozási szokások ill. eszközök. Így pl. minden X alkalmazásnak indítási paraméterként megadható az induló ablak mérete, színe, pozíciója stb. Talán a legfontosabb indítási paraméter a display neve, amelyen az adott alkalmazás megjelenik. Ez vagy a **-display** kapcsoló után adható meg, vagy

[26] Fontos, hogy ez végrehajtható legyen, mert a /usr/lib/X11/Xsession script exec paranccsal hajtja végre a .xsession-t. Ezzel szemben a .xinitrc-t egy shell hajtja végre így nem szükséges végrehajthatónak lennie.

ennek hiányában a **DISPLAY** környezeti változóban. A display neve három részből tevődik össze:

```
host_név:szerver_sorszám.képernyő_sorszám
```

A host név annak a számítógépnek ill. berendezésnek a hálózati a neve, amelyen a szerver program fut. A szerver sorszám az azonos gépen futó szerver programok megkülönböztetésére szolgál és 0 az első. A képernyő sorszám pedig a szerver által kezelt képernyő sorszáma. Szintén 0 az első de ez elmaradhat. A következő példában az xclock programot indítjuk el az aida.fsz.bme.hu gép 0. szerverének 0. képernyőjére 200x300-as méretben.:

```
xclock -display aida.fsz.bme.hu:0 -geometry 200x300
```

Nem szükséges külön megadni a display nevét, ha a DISPLAY környezeti változó be van állítva. Ez csh típusú (csh, tcsh) shell esetén a setenv paranccsal, sh típusú shell esetén (sh, bash) pedig az export paranccsal lehetséges. Pl.

```
setenv DISPLAY aida.fsz.bme.hu:0.0
```

vagy

```
DISPLAY = aida.fsz.bme.hu:0.0
export DISPLAY
```

A DISPLAY környezeti változót rendszerint az xinit, vagy az xdm beállítja a szerver nevének megfelelően. Abban az esetben azonban, amikor egy helyi számítógép grafikus felületén keresztül egy másik számítógépre jelentkezünk be (pl. telnet, vagy rlogin segítségével), akkor a távoli gép DISPLAY változója nem állítódik be. Ilyenkor ezt kézzel (vagy trükkös login script-tel) kell beállítani. Ez azonban nem elég ahhoz, hogy a távoli gépen indított kliens program a helyi X szerverhez kapcsolódhasson. Rendszerint ugyanis biztonsági okokból a munkaállomások szerverei úgy vannak konfigurálva, hogy csak velük azonos gépen futó klienseket hajlandóak kiszolgálni. Egészen pontosan egy adatbázis tartalmazza azoknak a gépeknek a nevét, amelyek kapcsolatba léphetnek a szerverrel. Ezt az adatbázist az **xhost** program segítségével lehet módosítani. Így pl az

```
xhost +host
```

engedélyezi

```
xhost -host
```

pedig megtiltja a host nevű gépen futó kliensek számára a kapcsolatfelvételt.

Az ablakkezelő program indítása a többi kliens program indításával azonos, hiszen az is egy kliens. Sokszor a megfelelő indító (startup) állományok automatikusan elindítják (pl. **xinitrc)**, de külön paranccsal is indítható. A kiválasztott ablakkezelő programot célszerűen leválasztva (háttérben) kell elindítani, mert másként nem kapnánk vissza a shell prompt-ját. Pl:

```
twm &
```

Az ablakkezelő elindulását az ablakok keretezésének megváltozása és rendszerint egy rövid hangjelzés is jelzi. Tanulságos kísérlet lehet az ablakkezelő munka közbeni leállítása, és egy másik ablakkezelő elindítása: Indítsunk el néhány klienst (xterm, xclock, xeyes, stb.) párhuzamosan és egy ablakkezelőt, ha még az nem futott. Ezután a kill paranccsal állítsuk meg az ablakkezelőt. Ekkor azonnal szembetűnik az ablakkezelő hiánya: Az ablakok nem mozgathatóak, nem változtatható a méretük, megváltozik a keretezésük stb. Ezután egy másik ablakkezelőt elindítva ismételten mozgathatóvá, újraméretezhetővé válnak az ablakaink.

A kliensek indításánál láttuk, hogy bizonyos indítási paramétereket a kliens programok egységes formában kezelnek, és ezzel a kliens megjelenése a felhasználó által módosítható ill. befolyásolható. Az X rendszer a felhasználói kívánságok ill. előírások (felhasználói preferenciák) megadására a következő módszereket definiálja:

- programhoz tartozó leíró állományok (app-defaults, XAPPLRESDIR)
- login katalógusban elhelyezett **.Xdefaults** állományok
- gyökér ablakhoz tartozó erőforrás adatbázis
- indítási paraméterek

A különböző módon megadott értékeket egy közös adabázisba gyűjti a rendszer, és az Xlib ún. erőforrás kezelő[27] (resource manager) rutinjaival kezeli, így biztosítja azok egységes kezelését.

A programhoz tartozó leíró állományok a /usr/lib/X11/app-defaults katalógusban a programmal azonos néven találhatók (pl: /usr/lib/X11/app-defaults/xterm). Ezt az állományt rendszerint az alkalmazás készítője készíti el, és alkalmazás részét képezi. Ennek módosításával a rendszergazda befolyásolni tudja egy-egy alkalmazás általános megjelenését ill. viselkedését.

A login katalógusban elhelyezett .Xdefaults állomány segítségével maga a felhasználó tudja saját preferenciáit megadni. Hasonlóan a felhasználó egyéni preferenciáit tartalmazza a gyökér ablakhoz tartozó adatbázis is, amelybe az **xrdb** adabáziskezelővel lehetséges adatokat bevinni.

Az egyes preferencia leírók feldolgozási sorrendje olyan, hogy az indítási paraméterekkel megadott érték prioritása a legnagyobb. Ezt követi a gyökér ablak adatbázisa, amit rendszerint a .Xdefaults és/vagy a .Xresources állományból tölt fel a szerver indító scriptje. Leggyengébb prioritással az app-defaults állományok szerepelnek.

Hálózati állományrendszer (NFS) használatakor lehetőségünk van a login katalógusban elhelyezett .Xdefaults állományok gépenkénti kialakítására. Ezt úgy érhetjük el, hogy `.Xdefaults-host` alakú állományneveket használunk, ahol a host az adott gép neve.

[27] Az X rendszer több különböző fogalmat is erőforrásnak nevez. Az itt használt erőforrás fogalom nem azonos a szervernél használt erőforrással.

Az erőforrások megadása igen egyszerű szintaxissal történik, amit egy minta .Xdefaults állományon keresztül mutatunk be:

```
xterm.VT100.scrollBar:    true
xterm*background:         red
xclock.clock.hands:       blue
```

Az erőforrást az alkalmazás neve, és az erőforrás megnevezése azonosítja. Ezt követi az erőforráshoz előírt preferencia értéke. Lehetőség van szabad paraméter megadására is a csillag karakter segítségével. A fenti példában az `xterm` alkalmazás `VT100` nevű objektumának `scrollbar` erőforrását bekapcsoltuk, és az alkalmazás összes ablakobjektumának piros háttérszínt adtunk. Az `xclock` esetében pedig a mutatók színét kékre állítottuk.
Azt, hogy egy alkalmazás milyen erőforrásokkal rendelkezik, és azok milyen értékeket vehetnek fel, az adott alkalmazás felhasználói leírásából tudhatjuk meg. Sokszor ezen leírások hivatkoznak az adott alkalmazást felépítő ún. widget-készlet erőforrásaira is. Ezek állítása rendszerint már mélyebb X programozói ismereteket igényel.

9.5 Kapcsolódó parancsok

A fontosabb hálózattal kapcsolatos parancsok: (A zárójelben szereplő számok a UNIX dokumentáció fejezetszámai.)

X(1) - Rendszerint az alapértelmezés szerinti X szerver program neve, de ettől fontosabb, hogy a man X hatására átfogó leírást kapunk az X rendszerről.
xinit(1) - az X szerver indító programja
twm,fvwm,fvwm2,
olvm,mwm
vuewm(1) - ablakkezelő programok
xhost(1) - hozzáférési jogokat szabályozó program
xterm(1) - alfanumerikus terminál emuláció
xmodmap(1) - billentyűzet eseménykódjait módosító program
xrdb(1) - X szerver erőforráskezelő segédprogramja
xbiff(1) - elektronikus levél érkezését figyeli
xcalc(1) - kalkulátor program
xclock(1) - analóg/digitális óra
xconsole(1) - konzol üzenetek megjelenítője
xdm(1) - XDMCP daemon
xeyes(1) - érdekes demo program
fs(1) - font szerver

C. Függelék: Shell-ek

A következőkben rövid áttekintést adunk a Bourne shell, és a C shell lehetőségeiről, programozásáról.

C.1 A Bourne Shell

Parancsállomány indítása:

sh állomány [argumentum...]

vagy ha az állomány végrehajtható (van x bit), akkor

állomány [argumentum...]

Speciális jelek:

*** ? [ ] < > | & $ { } ; ( )**

**' " ` **

Speciális jelentés megszüntetése:

	'	"	`	\	$	*
'	t	n	n	n	n	n
"	n	t	i	i	i	n
`	i	i	t	i	i	i

t terminális
n nem értelmeződik speciálisan
i értelmeződik

Shell változók:

```
alma=kicsi    szilva=nagy    pari=piros
echo $alma $szilva ${pari}ka
```

Speciális változók:

$?	exit státusz
$0	parancsállomány neve
$#	indítási paraméterek száma
$$	processz szám
$!	utolsó háttérben futtatott proc. száma
$-	aktuális shell kapcsolók
$*	az összes indítási paraméter
$@	ua. min $*, ha nincs "" között "$*" == "$1 $2...", de "$@" == "$1" "$2"...
$HOME	home katalógus neve
$CDPATH	keresési út a cd parancshoz
$PATH	keresési út a programok indításához
$PS1	elsődleges prompt
$PS2	másodlagos prompt
$IFS	input mező határoló karakter

Paraméterhelyettesítés

${v-d}	d, ha v nincs beállítva
${v=d}	ua. mint az előző, de v-t is beállítja
${v?message}	message kiírása és megáll, ha v nincs beállítva

A -, =, vagy ? előtt : jelet használva nem v létezését teszteli, hanem azt, hogy az **üres** vagy **nem üres**. Pl **${v:=d}.**

Parancshelyettesítés

Egy parancs eredményét egy másik parancs paraméterként értelmezhet.

```
dir=`pwd`; echo $dir
```

Állománynév-helyettesítés

*****	tetszőleges számú tetszőleges karakter
?	pontosan egy tetszőleges karakter
[abc]	**a** vagy **b** vagy **c** karakter egyike
[m-n]	**m**-től n-ig intervallumból egy karakter

I/O átirányítás

>állomány	standard output az állományba
>>állomány	standard output az állományhoz fűződik
<állomány	standard input az állományból
<<word	standard input a parancsállomány a word-ot tartalmazó sorig
>& n	a standard output helyett az n. állományleírót használja
<& n	a standard input helyett az n. állományleírót használja

A fenti jelölések bármelyikét megelőzheti egy szám. Ekkor a standard input ill. output helyett a számnak megfelelő állományleírót használja. Pl:

2>állomány	a standard error-t írja az állományba
2>&1	a standard error és a standard output összekapcsolódik

p1 | p2	csővezeték szervezése (p2 a standard inputot a p1 standard outputjából veszi)

Beépített parancsok:

:	ne tégy semmit
. állomány	include
break[n]	ciklus elhagyása
continue[n]	következő ciklusra
cd[arg]	change directory
echo[arg]	echo
eval[arg]	arg végrehajtása (arg mint input)
exec[arg]	arg végrehajtása, de nem indul új shell
exit[n]	megállás n státusszal
export[var]	var exportálása
hash[-r][nam]	hash tábla újragenerálása
pwd	az aktuális munkakatalógus kiírása

read var...	var változóba olvas a standard inputról
readonly[var]	var nem változtatható tovább
return[n]	visszatérés függvényből
set[kapcs[arg]]	kapcsolók beállítása
shift[n]	pozícionális paraméterek léptetése
test	feltétel kiértékelése
times	futási idők
trap[arg][n]	szignál kezelés
type[arg]	hogyan értelmezné a shell az arg-ot
ulimit[-fp][n]	állománylimit beállítás
umask[ddd]	default állományattribútumok
unset[var]	változó megszüntetése
wait[n]	várakozás az n. processzre

Feltétel kiértékelése (test):

A programok visszatérési státusza alapján elágazásokat lehet szervezni. A 0 az igaz, a nem 0 a hamis érték.

test s	igaz, ha s nem null string
test -z s	igaz, ha s nulla hosszúságú string
test -n s	igaz, ha s nem nulla hosszúságú string
test s1 = s2	igaz, ha s1 string egyezik s2 stringgel
test s1 != s2	igaz, ha s1 string nem egyezik s2 stringgel
test n1 -eq n2	igaz, ha n1 aritmetikailag egyenlő n2-vel. **-eq** helyett **-ne, -le, -lt, -ge, -gt** is használható a megfelelő feltétel megfogalmazásához.
test -f f	igaz, ha f létezik, és nem katalógus
test -r f	igaz, ha f olvasható
test -w f	igaz, ha f írható
test -d f	igaz, ha f katalógus
test -s f	igaz, ha f létezik, és nem nulla hosszúságú
test -t fd	igaz, ha fd egy megnyitott file filedscriptora, és az terminál.

A fenti műveletek kombinálhatók a **!** (tagadás), **-o** (vagy), **-a** (és) logikai műveletekkel valamint zárójelezhetők.

Vezérlési szerkezetek:

If feltétel:

```
if parancsok
  then parancsok
  else parancsok
fi
```

```
if test $1 = 'alma'
   then echo '$1=alma'
fi
```

For ciklus:

```
for i in w1 w2 ...
  do parancsok
done
```

```
for i do grep $i /etc/passwd
done
```

```
for i in alma körte
   do echo $i
done
```

While cilkus:

while parancsok	**while** test $1
do parancsok	**do** echo $1; shift
done	**done**

Until cilkus:

until parancsok	**until** test $1 = ''
do parancsok	**do** echo $1; shift
done	**done**

Case szerkezet:

case word **in**	**case** $# **in**
minta1) parancsok ;;	**1)** echo 1 ;;
minta2) parancsok ;;	**2)** echo 2 ;;
...	***)** echo sok
esac	**esac**

Parancssorozat:

Az egyes parancsok parancssorozatba rendezhetők a ';' '||' és az '&&' jelek segítségével. Míg a ';' egyszerű elválasztóként szerepel, addig a '||' és '&&' jelek jelentése azonos a C nyelvben megszokottal. Így pl a

```
parancs1 || parancs2
```

sorozatból a parancs2 csak akkor fog végrehajtódni, ha a parancs1 hamis (azaz nem nulla) megállási státusszal állt meg. A

```
parancs3 && parancs4
```

sorozatból pedig a parancs4 csak akkor fog végrehajtódni, ha a parancs3 igaz (azaz nulla) megállási státusszal állt meg.

Példák:

1. Írjuk ki 1-25-ig az egész számokat:

```
i=1
while [ $i -le 25 ]
do
        echo $i
        i=`expr $i + 1`
done
```

2. Írjuk ki hogy hány file van a katalógusban:

```
n=0
for i in *
do
        if [ -f $i ]; then
                n=`expr $n + 1`
```

```
        fi
    done
    echo $n file van a katalogusban
```

3. Telefonregiszter shell script program:

```
    #!/bin/sh
    #Telefonregister (adatfelvetel)
    #Hasznalat: newtel
    date=`date`
    while   echo -n "NAME: "
        read name
        do      case $name in
            ")      continue ;;
            q)      break ;;
            esac
        echo -n "ADDRESS: "
        address=
        while read a
        do      case $a in
            ")      # blank line ends address
                break ;;
            *)      if [ "x$address" = "x" ] ; then
                  address="$a"
                else
                  address="$address~$a"
                fi ;;
            esac
        done
        echo -n "PHONE: "
        read phone || break
  echo "$name:$phone:$address:$date">>$HOME/.phones
    done
```

4. Telefonregiszter lekérdező script:

```
    #!/bin/sh
    #Telefonregister (lekerdezo)
    #Hasznalat: tel lekerdezo_string
    if [ ! -r $HOME/.phones ] ; then
        echo "No phone database"
        exit
    fi
    for i do
        grep -i "$i" <$HOME/.phones | awk '
        BEGIN {
            FS=":"
        }
        {
  printf("%-25.25s %12.12s   %-38.38s\n", $1, $2, $3)
        }'
```

```
done
```

C.2 A C Shell

Parancsállomány indítása:

csh állomány [argumentum...]
vagy ha az állomány végrehajtható (van x bit), akkor
állomány [argumentum...]

Speciális jelek:
(szinte minden ami, nem betű és nem szám)

*** ? [ ] < > | & $ { } ; ( ) ! ~ : ^ @ % #**

**' ' " " ` ` **

Speciális jelentés megszüntetése:

	'	"	`	\	$	*	!	~
'	t	n	n	n	n	n	i	n
"	n	t	i	i	i	n	i	n
`	i	i	t	i	i	i	i	i

t terminális
n nem értelmeződik speciálisan
i értelmeződik

History és history-helyettesítés:

Az utolsó n parancs egy eseménytárba kerül, ahonnan ismét elő lehet venni azokat, vagy fel lehet használni egyes részüket új parancs kialakításához.

A **!** segítségével lehetséges egy korábbi parancssorra hivatkozni. A felkiáltójel csak akkor helyettesítődik speciális értékkel, ha **betű**, **@** vagy **számjegy** követi.
History hivatkozások:

!n	n. parancssor
!-n	n. parancssor visszafelé (relatív)
!minta	mintára illeszkedő parancssor
!!	utolsó parancssor

Szó választása a kijelölt sorból **kettősponttal** lehetséges (a **:** elhagyható, ha a szelektor nem betű vagy nem szám):

n	n. szó (0. a legelső)
^	első paraméter (1. szó)
$	utolsó szó
%	a mintára illeszkedő szó
m-n	kiválasztott szavak
-n	ua. mint **0-n**
*****	ua. mint **^-$**
m*	ua. mint **m-$**

m-	ua. mint **m***, de **$** nincs benne

Az így kijelölt szó vagy szavak a következő módosítókkal módosíthatók:

h	eltávolítja az útnévből a végét (head)
t	eltávolítja az útnévből az elejét (tail)
r	eltávolítja a kiterjesztést a névből
s/m/t/	**m** mintát **t** szöveggel helyettesíti
p	kiírja az új parancsot, de nem hajtja végre
&	az előző helyettesítést ismétli
g	globális csere

A helyettesítés megadásában / határoló jel helyett bármilyen más határoló is használható. A **t** szövegben a **&** jel az **m** mintával helyettesítődik. A speciális jelentés feloldására használhatók a \ **'** **"** jelek. A **"** jel esetében azonban a változók helyettesítése megtörténik.

Ha a history hivatkozás elmarad, akkor a szókijelölések ill. módosítók az utoljára illesztett parancssorra vonatkoznak, és nem az utolsó history sorra. (Pl. a **!?lo?:1 !$** az első és utolsó paramétert jelenti abból a sorból, amire a **?lo?** illeszkedik.)

Alias és alias-helyettesítés:

A shell minden parancssor első szavát megpróbálja az **alias** listájára illeszteni. Ha talált illeszkedőt, akkor az alias-hoz tartozó szöveget a history-helyettesítés szabályai szerint értelmezi. Így az aktuális parancssorra a felkiáltójellel lehet hivatkozni, aminek egyes részletei a history-helyettesítésnél megismert módon használhatók fel. Pl. !:5 az aktualis parancssor 5. szavát jelenti. (Figyelem a felkiáltójel speciális jelentését időlegesen meg kell szüntetni pl: **\!:5**, így az az alias végrehajtásakor jut csak érvényre.)

Változók és változóhelyettesítés:

Az alias-helyettesítés után a shell helyettesíti a változókat, amelyek **$** jellel kezdődnek. A **@** jel hatására a változót numerikusan értékeli ki.

```
set alma=(kicsi csacsi) szilva="nagy kek" pari=piros
echo $alma $szilva ${pari}ka $alma[1] $szilva[1]
```

Zárójelek alkalmazásával ún. szólisták alkothatók, amelyekből szelektorok segítségével szavak választhatók ki pl: **$var[n]**, ahol n egy szám, vagy intervallum. (1 jelenti az első szót)

Speciális változók:

$#var	var szavainak száma
$n	n. indítási paraméter
$#	indítási paraméterek száma
$$	processz szám
$*	az összes indítási paraméter
$?var	1 ha var létezik, 0 ha nem
$<	standard inputról olvasott sor (beolvasás)
$argv	indítási paraméterek

$cdpath	cd keresési útvonala
$child	utoljára leválasztva elindított processz
$cwd	aktuális munkakatalógus útneve
$history	history mérete
$home	home katalógus neve
$ignoreeof	nem áll le az állomány végénél
$noclobber	átirányításnál figyelmeztet, ha az állomány létezik
$filec	állománynév-folytatás engedélyezése
$fignore	állománynév-folytatásból kizárt kiterjesztések listája (pl. set fignore = (.o .out)
$status	utolsó parancs státusza
$path	keresési út a programok indításához
$prompt	prompt string

Parancshelyettesítés

Egy parancs eredményét egy másik parancs paraméterként értelmezhet.

mail `**cat címek**` <level

Állománynév-helyettesítés

*****	tetszőleges számú tetszőleges karakter
?	pontosan egy tetszőleges karakter
[abc]	**a** vagy **b** vagy **c** karakter egyike
[a,b,c]	ua. mint **[abc]**
[m-n]	**m**-től n-ig intervallumból egy karakter
~	login (home) katalógus útneve
~user	**user** felhasználó login katalógusa

A ~ csak szó elején helyettesítődik speciálisan.

I/O átirányítás

>állomány	standard output az állományba
>! állomány	standard output az állományba (ha a **noclobber** változó be van állítva, és az állomány létezik, akkor nincs hibajelzés)
>&állomány	standard output + error az állományba
>>állomány	standard output az állományhoz fűződik
>>! állomány	standard output az állományhoz fűződik (ha a **noclobber** változó be van állítva, és az állomány nem létezik, akkor nincs hibajelzés)
<állomány	standard input az állományból
<<word	standard input maga a parancsállomány a word-ot tartalmazó sorig
p1 | p2	csővezeték szervezése (p2 a standard inputot a p1 standard outputjából veszi)
|&	a standard error és a standard output pipe-ba

Beépített parancsok:

#megjegyzés	megjegyzés (kivéve a file első sorában (#!/bin/csh))
alias név	

alias név lista	Alias lista kiíratása, bővítése
bg	
bg %job	job a háttérbe
break	foreach vagy while elhagyása
cd	
cd arg	change directory
continue	foreach vagy while ciklus folytatása
echo arg	echo
eval arg	arg végrehajtása, mint input
exec arg	arg végrehajtása, de nem indul új shell
fg	
fg %job	job az előtérbe
exit n	megállás n státusszal
goto cimke	ugrás a **cimke:** sorra
history	history kiíratása
kill -n %job	signál küldése a job processzeinek
kill -n pid	signál küldése a pid processznek
limit	erőforrás limit kiíratása, beállítása
notify	job státusz változásról értesítés
onintr -	
onintr cimke	megszakítás kezelés
rehash	hash tábla újraépítése
set	
set var	
set var=szó	
set var[index]=szó	
set var=list	változók kiíratása, értékadás
setenv var érték	környezeti változó beállítása
shift	
shift var	(pozícionális) paraméterek léptetése
source var	a parancs input a var lesz
stop	
stop %job	job leállítás
suspend	a shell átmeneti megállítása
umask	
umask ooo	default állományattribútumok
unalias minta	alias(ok) megszüntetése
unset minta	változó(k) megszüntetése
unsetenv minta	környezeti változó(ok) megszüntetése
wait	várakozás a gyerek processzekre
%job	előtérbe kerül a job
%job &	háttérbe kerül a job
@	
@ var[index]= kif	
@ var = kif	változók kiíratása, értékadás. A @ engedélyezi a változó numerikus kiértékelését.

Kifejezések:

Néhány beépített parancs (**@**, **exit**, **if**, **while**) paramétereként kifejezést (**kif)** kell megadni. Ezen kifejezésekben a C nyelvben használatos operátorok alkalmazhatók:

|| && | ^ == != <= >= < > << >> + - * / % ! ~ ( )
++ -- += -= stb

Egy parancs végrehajtása és annak visszatérési státusza is szerepelhet a kifejezésben. Ilyenkor a végrehajtandó parancsot **{ }** jelek közé kell tenni. Pl:

@ i = { grep -s ubul $i }

Egy program végrehajtása akkor sikeres, azaz igaz (1), ha a program megállási státusza 0. Egyéb esetben a végrehajtás hamis értékű (0).

Az állományok egyes tulajdonságait **-q állománynév** alakú lekérdező kifejezéssel vizsgálhatjuk meg, ahol **q**:

r	igaz, ha az állomány olvasható
w	igaz, ha az állomány írható
x	igaz, ha az állomány végrehajtható
e	igaz, ha az állomány létezik
o	igaz, ha az állomány tulajdonosa megegyezik a lekérdezővel
z	igaz, ha az állomány nulla hosszúságú
f	igaz, ha az állomány egyszerű állomány
d	igaz, ha az állomány katalógus

Vezérlési szerkezetek:

If feltétel:

if (kif**) then**
parancsok
else
parancsok
endif

```
if (-r alma) then
        echo olvashato
else
        echo nem olvashato
endif
```

Foreach ciklus:

foreach var **(**lista**)**
parancsok
end

```
foreach i (alma korte fa)
        echo $i
end
```

While cilkus:

while (kif**)**
parancsok
end

```
while ($i < 12)
        @ i++
        echo $i
end
```

Switch szerkezet:

```
switch (szó)
    case minta1:
        parancsok1
    breaksw
    case minta2:
        parancsok2
    breaksw
    ...
    default:
        parancsok3
    breaksw
endsw
```

```
switch ($#)
    case 1:
        echo egy
    breaksw
    case 2:
        echo ketto
    breaksw
    default:
        echo sok
    breaksw
endsw
```

Parancssorozat:

Az egyes parancsok parancssorozatba rendezhetők a ';' '||' és az '&&' jelek segítségével. Míg a ';' egyszerű elválasztóként szerepel, addig a '||' és '&&' jelek jelentése azonos a C nyelvben megszokottal. Így pl a

```
parancs1 || parancs2
```

sorozatból a parancs2 csak akkor fog végrehajtódni, ha a parancs1 hamis (azaz nem nulla) megállási státusszal állt meg. A

```
parancs3 && parancs4
```

sorozatból pedig a parancs4 csak akkor fog végrehajtódni, ha a parancs3 igaz (azaz nulla) megállási státusszal állt meg.

Példák:

1. Prompt string átalakítása MS-DOS pg formára:
   alias cd 'cd \!*; set prompt=$cwd\>'

2. Készítsünk aliast egy tetszőleges file rendezésére és egy masik állománnyal való összehasonlítására:
   alias has 'sort cd \!*; set prompt=$cwd\>'

C.3 A shell-ek beállítása, indulás, megállás

A shell-ek különböző beállítása, számtalan lehetőséget ad a felhasználóknak, hogy az egyéni ízlésüknek, munkamódszerüknek megfelelő környezetet kialakítsák. Ezen beállítások egy része rendszer specifikus, és rendszerint a rendszergazda adja meg, más része viszont teljesen egyéni. A különböző shell változatok más-más lehetőséget biztosítanak ezen beállítások megadására, egy azonban közös vonás, hogy shell script-ek végrehajtódásának eredményeként jönnek létre. A következőkben röviden összefoglaljuk a legelterjedtebb shell-ek ezen mechanizmusait.

A beállítások szempontjából négy jól definiált pont határozható meg az shell-ek működésében:

1. indulás mint login shell (login)
2. indulás mint nem login shell (start)
3. megállás mint login shell (logout)
4. indulás mint nem interaktív shell (rsh)

Ennek megfelelően különböző shell script-ek hajtódnak végre:

shell	login	start	logout	rsh
sh	/etc/profile $HOME/.profile			
ksh	/etc/profile $HOME/.profile			
csh	/etc/csh.cshrc /etc/csh.login ~/.cshrc ~/.login	/etc/csh.cshrc ~/.cshrc	/etc/csh.logout ~/.logout	/etc/csh.cshrc ~/.cshrc
tcsh	/etc/csh.cshrc /etc/csh.login ~/.tcshrc || ~/.cshrc ~/.login	/etc/csh.cshrc ~/.tcshrc || ~/.cshrc	/etc/csh.logout ~/.logout	/etc/csh.cshrc ~/.cshrc
bash	/etc/profile ~/.bash_profile ||~/.bash_login || ~/.profile	~/.bashrc	~/.bash_logout	. $ENV

A táblázatban || jellel jelöltük a vagy kapcsolatot. Ezekben az esetekben csak az a script hajtódik végre, amelyiket a megadott sorrendben keresve megtalálja az adott shell.

Megjegyezzük, hogy egyes C shell változatok a nem hajtják végre a cshrc script-et a login fázisban, illetve egyes változatok viszont előbb hajtják végre a login script-et.

Egyes rendszerekben a /etc/csh.cshrc, /etc/csh.login és a /etc/csrc.logout állomány helyett a /etc/cshrc, /etc/login és /etc/logout, vagy a /etc/cshrc.std, /etc/login.std és /etc/logout.std állományok találhatók.

E. Függelék: Szövegszerkesztők (editorok)

Szinte minden felhasználó által a legtöbbet használt programok a szövegszerkesztők. Éppen ezért talán ezen programokból van a legtöbb a világon. Arról, hogy melyik a jobb és melyik a rosszabb, értelmetlen lenne beszélni, hiszen a megoldandó feladatok igen sokfélék, és lehet, hogy ami egyik feladathoz jobb, az a másik feladathoz alkalmatlanabb. A UNIX rendszerekben is sok különböző editor program terjedt el, amelyek között van néhány, a UNIX rendszerrel egyidős program is (pl. **ed, sed**). Mivel ezek manapság is használatos programok, ismeretük úgymond az alapokhoz tartozik.

E.1 Az ed editor

A UNIX rendszerek mindegyikében megtalálható az **ed** editor. Ez az editor az MS-DOS-ban használatos különböző editoroktól sokban eltér. Ezen eltérések közül talán a legszembetűnőbb, hogy használata minden bizonnyal kényelmetlenebb, mégis elterjedten használt eszköz a UNIX-os környezetben. A UNIX rendszerek első (1970-es évek elején készült), és még mai napig használt editora az ed editor. Ez egy sor és orientált editor, amely a UNIX keletkezésekor általánosan használt írógépszerű, papírra író megjelenítőkön lehetővé tette viszonylag kényelmesen a különböző szövegállományok létrehozását, javítását. Ma már szinte kizárólag képernyő-orientált megjelenítőket használnak, amelyen az ed használata értelmetlenül kényelmetlennek tűnik. Valójában jelentőségét az adja, hogy

- igen egyszerű parancskészlete miatt könnyen megtanulható,
- hajlott kora ellenére néhány dolgot egészen jól csinál,
- minden rendszerben egy biztos pontnak számít, mivel minden mai UNIX is implementálja az ed programot, így nem érheti az embert meglepetés,
- az ed programmal, és a szinte azonos parancskészlettel rendelkező **sed** (stream editor) programmal parancsállományból is lehetséges a szövegek javítása.
- a viszonylag elterjedt **vi** editor is megvalósítja az ed parancsokat,
- egyszerű parancskészletét felhasználva a **diff** program képes olyan javító sorozatot előállítani, amellyel az egyik állományból automatikusan elállítható a módosított állomány.

Az ed a bemeneti állományt egy ún. editpufferbe olvassa be, és minden műveletet ebben hajt végre. A program indítása után parancsra vár (amit semmivel nem jelez). A parancsok egybetűs kódból állnak, amit megelőzhet egy vagy két ún. sorcím, ami kijelöli, hogy mely sorokra ill. sortól kezdődően hajtsa végre az adott parancsot. Ennek hiányában a parancs alapértelmezés szerint hajtódik végre. Egy sorba csak egy parancs írható, kivéve a p, és az l parancsokat, amelyeket a legtöbb parancs után is megadhatunk. Ha valami szintaktikai hibát követünk el, akkor nem túl bőbeszédűen ezt egy "?" kiírásával jelzi, majd újabb parancs megadására vár az ed. Minden más hibánál is ezt az "hibaüzenetet" kapjuk. Pl. ha az editpuffert még nem írtuk ki, de egy újabb állományt kezdünk javítani az e paranccsal. Ilyenkor a parancs megismétlésekor már nem jelez hibát.

Lássuk először a sorcímeket:

n Az n sorszámú sor, ahol n egy pozitív egész szám.

$ Az utolsó sor.

. Az aktuális sor.

/minta/ A következő "minta"-val egyező sor. Az állomány végén a keresés az állomány elejétől folytatódik.

?minta? A megelőző "minta"-val egyező sor. Az állomány elején a keresés az állomány végétől folytatódik.

'c A c-vel megjelölt sor.

A "minta"-t a már megismert reguláris kifejezések használatával adhatjuk meg. A sorcímeket "," (vessző) vagy ";" (pontosvessző) választhatja el egymástól. A pontosvessző abban különbözik a vesszőtől, hogy ekkor az első sorcím meghatározása után a sormutató (.) ezt az értéket veszi fel, és csak ezután értékeli ki a második sorcímet. Ha egy parancsnak több címet adunk meg, mint amennyit az vár, akkor csak az utolsó vagy az utolsó kettő sorcímet használja fel. A továbbiakban a sorcímek helyett X és Y jelet is használunk:

X±n Az (X ± n) sorszámú sor.

X± Az (X ± 1) sorszámú sor. Ha több +, vagy - karakter van akkor azok hatása összeadódik.(pl. X-- azonos X-2-vel);

X,Y X -tól és Y-ig tartó sorok, beleértve a határokat is.

X;Y Az aktuális sort (.) X-re állítja, majd kiszámítja Y-t.

A következőkben a parancsokat úgy adtuk meg, hogy amelyek előtt sorcím is megadható, ott feltüntettük azok alapértelmezés szerinti értéket. Így ebből kiolvasható hogy, hány címet értelmez a parancs, és a hogy mi történik, ha sorcím nélkül használjuk a parancsot.

.a
szöveg
. Szöveg bevitele az aktuális sor után (append). A szöveg végét sor elején álló "." (pont) jelzi.

.,.c
szöveg
. A megadott sorok cseréje új szövegre (change). Az új szöveg vége ugyanaz, mint az a parancsnál.

.,.d A megadott sorok törlése.

e file A file nevű állomány beolvasása az edit pufferbe. Az eddigi puffertartalom felülíródik. (Ha nem írtuk ki, elveszik!)

E file Megegyezik az e paranccsal, de nem ad figyelmeztető üzenetet, ha az edit puffert még nem írtuk ki.

f file Az aktuális állománynév "file" lesz.

1,$g/minta/parancsok
A megadott tartományban a mintára illeszkedő sorokon a "parancsok" végrehajtása. (Több parancsot is megadhatunk \újsor-ral elválasztva.) Az egyes sorokon a parancsok úgy hajtódnak végre, hogy a végrehatás előtt az aktuális sor címe kerül a sormutatóba (.).

.i
szöveg

. Szöveg bevitele az aktuális sor elé (insert). A szöveg vége ugyanaz, mint az a parancsnál.

.,.+1j A megadott sorok egybefűzése.

.kc A sor megjelölése a c betűvel. (c csak kisbetű lehet.)

.,.l Sorok kiírása a nem látható karakterek kijelzésével.

.,mX A megadott sorok átvitele az X sor után.

.,.p A megadott sorok kiírása.

.,.P Ugyanaz, mint p.

q Kilépés a programból.

Q Ugyanaz, mint q, de nincs figyelmeztetés, ha az edit puffert még nem írtuk ki.

$r file A file nevű állomány beolvasása a megadott sor után. Ha nem adjuk meg a file paramétert, akkor az aktuális állományból olvas. Ha még nem volt aktuális állomány, akkor a file lesz az.

.,.s/minta/szöveg/
.,.s/minta/szöveg/g

A megadott sorokban a "minta" cseréje "szöveg"-re. Ha egy sorban sikerült a mintát megtalálni, akkor abban a sorban már nem illeszt újra, csak akkor, ha g módosítót megadtuk. A / jel helyett bármilyen más elválasztó karaktert is használhatunk. A "szöveg"-ben & jellel hivatkozhatunk a megtalált mintára, a \n sorozattal (ahol n egy számjegy), pedig a mintában levő n. \(r\) alakú kifejezésre.

.,tX A megadott sorokat az X sor után másolja.

u Az utoljára módosított sort visszaállítja a módosítás előtti állapotra. (Csak egy sort.)

1,$v/minta/parancsok

A megadott tartományban a mintára nem illeszkedő sorokon a "parancsok" végrehajtása.

1,$w file A megadott sorokat a file nevű állományba írja. Ha nem volt aktuális állománynév, akkor ez a file lesz. Ha paraméter nélkül adjuk meg, akkor az aktuális állományba ír.

1,$W file Megegyezik a w paranccsal. Ha létezett már az állomány akkor, azt nem törli, hanem a végéhez írja a kijelölt sorokat.

x Titkosító mód bekapcsolása.

$= A megadott sor szorszámának kiírása.

!parancssor

A parancssort átadja a shell-nek (sh-nak).

.+1,.+1 A sorcím parancs nélkül a megadott sorok kiírását jelenti.

Általában minden parancs az aktuális sort (.) az utolsó érintett sorra állítja, de az f, k, w, x = és ! parancsok nem változtatják meg.

Pl. indítsuk el az ed programot a következő paraméterrel:

```
ed /etc/passwd
```

Erre a program ki fogja írni a beolvasott karakterek számát, majd parancsra várakozik. A

```
1,$p
$
a
user19::123:77:Demo User:/tmp/user19:/bin/sh
.
w /tmp/cicus
q
```

Parancssorozat hatására először kiírásra kerül a /etc/passwd állomány összes sora (1,$p). Ezután az aktuális sor az utolsó sor lesz ($), majd ezután a sor után hozzáírjuk az állományhoz a

```
user19::123:77:Demo User:/tmp/user19:/bin/sh
```

sort. Ezt követően az eredményt kiírjuk a /tmp/cicus nevű állományba, majd kilépünk a programból.

E.2 A sed editor

A sed (stream ed) editor nem sokkal az ed után jelent meg a UNIX rendszerekben. Néhány feladat ugyanis megkívánta, hogy a standard bemeneten érkező szövegen nem interaktív módon egyszerű módosításokat végezzünk. A feladat elvégezhető az ed segítségével is, de az ed használatához a bemeneti szövegnek állományban kell lennie, ezért azt először egy ideiglenes állományba kell irányítani, és az ed program standard bemenetére adni a megfelelő parancsokat. A sed lényegében egy olyan ed, amely az első indítási paraméterét értelmezi a parancsok listájának, a javítandó szöveget a standard bemenetről vagy a megadott állományokból veszi, az eredményt pedig a standard kimenetre írja. A sed tehát az ed parancsait értelmezi. Van azonban néhány különbség ill. kiegészítés. A legfontosabb különbség, hogy a sed automatikusan minden bemeneti sort kiír, ha ezt nem tiltjuk le -n kapcsolóval indításkor. A másik fontos különbség, hogy a megadott parancsokat az összes bemeneti sorra végrehajtja, mintha az ed programnál minden parancsot g paranccsal adtunk volna meg. Természetesen ahogy a g parancsnál meg lehet adni feltételt a végrehajtáshoz, úgy a sed parancsai is végrehajthatók szelektíven. A parancsok végrehajtását a parancs elé írt sorcímek befolyásolják. Maximum két sorcím adható meg:

- Ha nem adtunk meg sorcímet, akkor a parancsot minden sorra végrehajtja.
- Ha egy sorcímet adunk meg, akkor a parancs csak azokra a sorokra hajtódik végre, amelyre a cím illeszkedik.
- Ha két sorcímet adunk meg, akkor azokra a sorokra hajtódik végre a parancs, amely az első sorcímre illeszkedő sor és a második sorcímre illeszkedő sor között (beleértve a határokat is) érkezik a bemenetről. A második sorcímre illeszkedő sor megérkezése után újból az első címet várja.

- A parancs elé írt "!" jel jelentése az. hogy pont azokra a sorokra hajtsa végre a parancsot, amelyre "!" nélkül nem hajtaná végre, és azokra ne hajtsa végre, amelyekre végrehajtaná.

A sorcímek hasonló módon írhatók elő, mint az ed programnál, de nincs megengedve a relatív hivatkozás a ± jellel. A reguláris kifejezések is ugyanúgy használhatók azzal a módosítással, hogy a \n (BACKSLASH kis n) sorozat az újsorjelet jelöli.

A sed egész pontosan úgy működik, hogy minden bemeneti sort egy ún. mintapufferbe olvas be. Ezután végrehajtja azokat a parancsokat, amelyek sorcímei illeszkednek. Ha a parancsok végére ért, akkor kiírja a mintapuffer tartalmát a standard kimenetre (ha nem volt megadva a -n kapcsoló). Ezután törli a mintapuffert.

A parancsok leírásánál zárójelben megadtuk a lehetséges sorcímek maximális számát. A szöveg argumentumok mindegyik sorát \ karakterrel kell zárni, kivéve az utolsót. Az így megadott szövegben a sor eleji szóközöket és tabulátorokat az első látható karakterig figyelmen kívül hagyja a program. Ahhoz, hogy a bevezető szóközök megmaradjanak, a \<SPACE> vagy a \<TAB> sorozatot kell használni.

(1).a
szöveg
: Szöveg kivitele a kimenetre az aktuális sor után (append). (Mielőtt a következő sort beolvasná.)

(2)b címke
: A megfelelő ":" parancsra való ugrás. Ha címke üres, akkor a sed script végére ugrik.

(2)c
szöveg
: A megadott sorok cseréje új szövegre (change). (A megfelelő sorokat törli a mintapufferből, és a kimenetre kiírja a szöveget.)

(2)d
: A megadott sorok törlése.

(2)D
: A mintapufferből csak az első sort törli.

(2)g
: A mintapuffer tartalmát a tároló puffer tartalmával helyettesíti.

(2)G
: A mintapuffer tartalmához hozzámásolja a tároló puffer tartalmát.

(2)h
: A tároló puffer tartalmát a mintapuffer tartalmával helyettesíti.

(2)H
: A tároló puffer tartalmához hozzámásolja a mintapuffer tartalmát.

(1)i
szöveg
: Szöveg kivitele az aktuális sor elé (insert).

(2)l
: A mintapuffer tartalmát a standard kimenetre így írja ki, hogy a nem látható ASCII karakterek láthatók legyenek.

(2)n
: A mintapuffer tartalmát a standard kimenetre kiírja, és beolvassa a következő bemeneti sort a mintapufferbe.

(2)N
: A mintapuffer tartalmához hozzámásolja a következő bemeneti sort.

(2)p
: A mintapuffer tartalmát a standard kimenetre kiírja.

(2)P
: A mintapufferből az első sort a standard kimenetre írja.

q
: Kilépés a programból.

Q Ugyanaz, mint q, de nincs figyelmeztetés, ha az edit puffert még nem írtuk ki.

(2)r file A file nevű állományt beolvassa és kiírja a standard kimenetre, mielőtt a következő bemeneti sort venné.

(2)s/minta/szöveg/módosító

A megadott sorokban a "minta" cseréje "szöveg"-re. Ha egy sorban sikerült a mintát megtalálni, akkor abban a sorban már nem illeszt újra, csak akkor, ha a g módosítót megadtuk. A / jel helyett bármilyen más elválasztó karaktert is használhatunk. A "szöveg"-ben & jellel hivatkozhatunk a megtalált mintára, a \n sorozattal (ahol n egy számjegy), pedig a mintában levő n. \(r\) alakú kifejezésre. A módosítók a következők lehetnek:

- **g** Újra illeszt.
- **p** Kiírja a mintapuffert a standard outputra.
- **w wfile** A mintapuffer tartalmát a wfile-hoz másolja.

(2)t címke Tesz. Ugrás a megadott címkére, ha volt helyettesítés a legutolsó bemeneti sor beolvasása vagy a legutolsó t utasítás végrehajtása óta.

(2)w file A mintapuffer tartalmát a wfile-hoz másolja.

(2)x Kicseréli a mintapuffer és a tároló puffer tartalmát.

(2)y/string1/string2/

Minden string1-ben előforduló karaktert a megfelelő srting2-beli karakterré konvertálja. A két stringnek azonos hosszúnak kell lenni.

(2)! parancs

Azokra a sorokra hajtja végre a parancsot, amelyek nem illeszkednek a megadott sorcímekre.

(0): címke Hatástalan, csak a címke megadására való.

(1)= Az aktuális sor sorszámát kiírja standard kimenetre.

(2){ A parancsok zárójelezésére szolgál. A megfelelő **}** jelig minden parancsot végrehajt.

(0) Az üres parancssor figyelmen kívül marad.

Példák:

1. Írjuk ki egy állomány összes sorát a 10. és a 20. sora között:

```
sed -n 10,20p
```

2. Töröljük ki egy állomány tizedik és huszadik sora között az összes olyan sort, ami az **alma** karaktersorozattal kezdődik.

```
sed -n '10,20{
        /^alma/d
        }'
```

E.3 A vi editor

A vi editor már a képernyő-orientált terminálokhoz készült. Nem mondható el róla, hogy nagyon kényelmes a használata, és még osztott képernyős üzemmódja sincs, mégis szinte ez a legelterjedtebben használt képernyő-orientált editor. Ha képernyő-orientáltan akarjuk használni, akkor ismernie kell a terminál típusát, amit a TERM környezeti változóból olvas ki. Ha a terminált nem tudja felismerni, akkor az csak ed üzemmódban használható. Ha sikerült a terminált felismernie, akkor

megjelenik a képernyőn a javítandó állomány egy része, és parancsmódban parancsra várakozik. Ekkor a kurzormozgató parancsokkal mozoghatunk az állományban, vagy éppen módot váltva javíthatunk, beszúrhatunk.

Kurzor mozgató parancsok:

⇐	**h**	**backspace**	balra	**b**	előző szóra
⇒	**l**	**space**	jobbra	**w**	következő szóra
⇓	**j**	+	le	**e**	szó végére
⇑	**k**	-	fel	**0**	sor elejére
fx	előre az x karakterre			**$**	sor végére
Fx	vissza az x karakterre			**;**	utolsó **fx** v. **Fx** ismétlése
nG	az n. sorra			**G**	az állomány utolsó sorára
(	előző mondatra			**{**	előző bekezdésre
)	következő mondatra			**}**	következő bekezdésre
H	képernyő első sorára			**M**	képernyő középső sorára
L	a képernyő utolsó sorára				
z(cr)	aktuális sor a kép tetejére			**z.**	aktuális sor a kép közepére
				z-	aktuális sor a kép aljára
/minta	minta keresése előre				
?minta	minta keresése hátra			**n**	utolsó keresés ismétlése

Javító parancsok:

a A	append mód a kurzor (**A**=sor vége) után (**ESC** végjelig)
i I	insert mód a kurzor (**I**=sor eleje) elé (**ESC**)
o O	open és append (**O**=insert) az aktuális sor után (**O**=elé) (**ESC**)
i(cr)	sor kettétörése
J	két sor egyesítése
R	replace mód (**ESC**)
r	egy karakter (**nr** n karakter) cseréje egy adott karakterre
s	egy karakter helyettesítése egy stringgel (**ESC**)
cw	a szó háralevő részének helyettesítése (**ESC**)
C	a sor hátralevő részének cseréje (**ESC**)
cc	sor cseréje (**ncc** n sor cseréje) (**ESC**)
.	az utolsó cserélő utasítás ismétlése
x	egy karakter törlése
dw	egy szó törlése
dd	egy sor törlése (**ndd** n sor törlése)
D	a sor hátralevő részének törlése
Y	egy sor (**nY** n sor) betöltése az általános bufferbe (yank)
p P	a korábban törölt vagy bufferbe töltött sorok kiírása az aktuális sor után (**P**=elé)
"bnY	n sor betöltése a 'b' bufferbe
"bp	a 'b' buffer tartalmának kiírása az aktuális sor után
"bP	a 'b' buffer tartalmának kiírása az aktuális sor elé

Egyéb parancsok:

u	undo az utoljára végrehajtott javító parancsra
^l	a képernyő tartalmának újrarajzolása
:w	az aktuális állomány kiírása az eredeti állományba
ZZ	az aktuális állomány kiírása az eredeti állományba és a vi megáll
:q	kilépés a vi programból
:q!	kilépés az állomány kiírása nélkül (editbuffer tartalma elveszik)
:n	a következő állomány javítása (pl. **vi *** esetén)
:!	shell behívása
:!cmd	'cmd'-t végrehajtja a shell
!!cmd	'cmd'-t végrehatja a shell, az eredmény az edit pufferbe kerül
:ed cmd	'ed' parancsmód, ahol cmd egy ed editor parancs.
:w name	az aktuális állomány kiírása a 'name' állományba
:e name	'name' állomány javítása
:e #	a következő állomány javítása (két állomány cserélődik)
:set list	kontroll karakterek megjelenítése
:set nolist	normál megjelenítés
:set number	sorszámozás bekapcsolása
:set nonumber	sorszámozás kikapcsolása
:set showmode	működési mód kijelzése
:set noshowmode	működési mód kijelzés kikapcsolás

Indítás:

vi +n állomány	állomány javítása az n. sortól
vi +/minta állomány	minta keresése az állományban és edit
vi + állomány	javítás az állomány utolsó sorában

M. Függelék: Parancsok leírásai

A következőkben a UNIX leírások formátumával azonos formátumban megadjuk néhány fontos parancs leírását. Azoknál a parancsoknál, ahol a parancs a UNIX két fő irányzatának megfelelő rendszerekben eltérő módon működik, megadtuk az általunk ismert eltéréseket is.

M.1 awk - riport generáló nyelv

Szintaxis

awk [-Fc] [prog | -f progfile] [állomány] ...

Leírás

Az awk a parancssorban megadott programot (prog), vagy a -f kacsoló után megadott állományban (progfile) levő programot hajtja végre. Inputkent a megadott állományokat, vagy a standard inputot tekinti. A nyelv részletesebb leírását ld. a 7.2. fejezetben.

BSD opciók

-Fc A mezőszeparátor a **c** karakter lesz.
-f Programot tartalmazó állomány megadása.

Lásd még

sh(1)

M.2 basename - útnév és kiterjesztés eltávolítása a névből

Szintaxis

basename string [suffix]

Leírás

A basename eltávolít minden előtagot (útnevet) és a megadott kiterjesztést (suffix) a stringből, majd kiírja a standard outputra.

Példák

A következő shell script a paraméterként kapott c programot leforítja, és az eredményt a program állomány kiterkjesztés nélküli nevére nevezi át.

```
cc $1
mv a.out `basename $1 .c`
```

Lásd még

sh(1)

M.3 cat - állományok összefűzése és kiíratása

BSD szintaxis

cat [-] [-benstuv] állomány ...

System V szintaxis

cat [-] [-estuv] állomány ...

Leírás

A cat sorra beolvassa mindegyik argumentumát, és kiírja a standard kimenetre. Ha bemeneti állománynevet nem adunk meg, vagy ez a "-" argumentum, akkor a cat a standard bemenetről olvas be.

BSD opciók

-b Sorszámokat ír a sorok elé, mint a -n, de az üres sorokat elhagyja
-e A nem látható karaktereket ugyanúgy írja ki, mint a -v, és minden sor végére egy "$" karaktert ír.
-n Sorszámokat ír a sorok elé.
-s Az egymás utáni üres sorokat egyetlen egy üres sorral helyettesíti.
-t A nem látható karaktereket ugyanúgy írja ki, mint a -v, és a TAB karakterek helyett pedig ^I -t ír.
-u A kimenetet nem puffereli.
-v A nem látható karaktereket is kiírja (kivéve a TAB, és az NL karaktert), úgy, hogy azok láthatók legyenek. A vezérlő karaktereket ^X alakban (CTRL-X), a nem ASCII karaktereket (amelyeknek a legmagasabb bitjük 1) pedig M-x alakban írja ki.

System V opciók

-e Ugyanaz, mint a BSD opciók.
-s Nem ad hibajelzést, ha egy állományt nem tud megnyitni.
-t Ugyanaz, mint a BSD, de a FORMFEED karaktereket ^L alakban írja ki.
-v Ugyanaz, mint a BSD, de a FORMFEED karaktert nem alakítja ^L alakra.

Példák

```
cat file1
cat file1 file2 > file3
```

Az első parancs kinyomtatja a file1 nevű állomány tartalmát a standard kimenetre, a második parancs pedig a file3 állományban egymás után elhelyezi a file1 és a file2 tartalmát, miközben a file1 és a `file2` változatlan marad.

Lásd még

pr(1), cp(1)

Megjegyzés

cat x y > x és cat x y > y parancsok hibás eredményre vezetnek, sőt az első esetben az "x" nevű, a második esetben pedig az "y" nevű állomány tartalmát nulla hosszúságúra csonkítják.

M.4 chgrp - állomány csoportjának megváltoztatása

Szintaxis

chgrp [-Rf] újcsoport állomány...

Leírás

A program minden megnevezett állomány csoport azonosítóját az **újcsoport** csoportra állítja. A csoport megadható számmal, vagy névvel is.

Csak az a tulajdonos, aki a megadott csoportba tartozik, vagy a super-user változtathatja megy egy állomány csoportját.

Opciók

-R Rekurzív végrehajtás a megadott katalógusok minden állományára. A szimbolikus linkek csoportját megváltoztatja, de nem követi a link-et.

-f Nem ad hibaüzenetet, ha nem sikerül a módosítás.

Lásd még

ls(1), chown(2), passwd(5), group(5)

M.5 chmod - állomány védelmi kódjának módosítása

Szintaxis

chmod [-Rf] mód állomány...

Leírás

A program minden megnevezett állomány védelmi kódját megváltoztatja a módban megadottak szerint. védelmi kódot nemcsak abszolút módon, hanem szimbolikusan is meg lehet adni.

Csak a tulajdonos, vagy a super-user változtathatja megy egy állomány védelmi kódjait.

Az abszolút kód egy oktális szám, ami a következő kódok vagy kapcsolatával állítható elő:

4000 setuid mód a végrehajtáskor
2000 setgid mód a végrehajtáskor
1000 sticky bit
0400 olvasás a tulajdonos által
0200 írás a tulajdonos által
0100 végrehajtás (keresés a katalógusban) a tulajdonos által
0070 olvasás, írás, végrehajtás (keresés) a csoport csoporttárs által
0007 olvasás, írás, végrehajtás (keresés) a többiek által

A mód szimbolikusan a következő módon adható meg:

[kinek] op jogok [op jogok] ...

Ahol a kinek a következő betűk kombinációja kehet:

u ha a az aktuális felhasználónak (user).
g ha a csoportnak
o ha a többieknek

Ha a kinek elmarad, akkor az mindenkit jelöl, de ilyenkor aktuális az umask-ot figyelembe veszi. Ha megadunk mindenkit (ugo), akkor az umask nem számít/

Az op +, -, vagy = lehet:

\+ ha jogokat adunk hozzá az állomány jogaihoz
\- ha jogokat veszünk el az állomány jogaiból
= ha pontosan ezt az értéket akarjuk beállítani

A jogok a következő betűk kombinációi lehetnek:

- r olvasási jog
- w írási jog
- x végrehajtási jog
- X végrehajtási jog, de csak akkor, ha a az állomány katalógus, vagy van már másik x bit-je.
- s setuid vagy setgid bit
- t stiky bit
- u a tulajdonos mezőt az eredeti módból veszi
- g csoporttárs mezőt az eredeti módból veszi
- o többiek mezőt az eredeti módból veszi

Opciók

- -R Rekurzív végrehajtás a megadott katalógusok minden állományára. A szimbolikus linkek védelmi kódja változatlan marad, és nem követi a link-et.
- -f Nem ad hibaüzenetet, ha nem sikerül a módosítás.

Példák

```
chmod g-w file
chmod +X file
```

Az első példa megtiltja az írást a csoporttársaknak.
A második példa végrehajthatóvá teszi az állományt mindenki számára, de csak akkor, ha már valaki által végrehajtható volt

Lásd még

ls(1), chmod(2), stat(2), umask(2), chown(8)

M.6 chown - állomány tulajdonosának megváltoztatása

Szintaxis

chgrp [-Rf] újtulaj állomány...

Leírás

A program minden megnevezett állomány tulajdonos azonosítóját az **újtulaj** tulajdonosra állítja. A csoport megadható számmal, vagy névvel is.

Csak a super-user változtathatja megy egy állomány tulajdonosát. Egyes UNIX változatokban azonban megengedett a tulajdonról való lemondás, azaz a tulajdonos is változtathat

Opciók

- -R Rekurzív végrehajtás a megadott katalógusok minden állományára. A szimbolikus linkek tulajdonosát megváltoztatja, de nem követi a link-et.
- -f Nem ad hibaüzenetet, ha nem sikerül a módosítás.

Lásd még

ls(1), chgrp(2), passwd(5), group(5)

M.7 cmp - két állomány összehasonlítása

Szintaxis

cmp [-ls] állomány1 állomány2

Leírás

A két állományt összehasonlítja. Ha az állomány1 helyén a "-" jel áll, akkor helyette a standard bemenetet használja. Az alapértelmezés szerinti opciók mellett a cmp nem jelzi, ha a két állomány megegyezik egymással. Ha különböznek, akkor megadja az állományok különbözővé válásának sorát, és byte-számát (vagyis annak a sornak és annak a byte-nak a számát, amelyik már nem egyezik meg). Ha az egyik állomány teljesen ugyanolyan, mint a másik, csupán előbb véget ér, ezen tényt is jelzi.

Opciók

-l Megadja a különbözőnek talált byte sorszámát az állományban (decimálisan), és a különböző byte-okat (oktálisan) minden egyes különbségnél.

-s Nem ír ki semmit különböző állományok esetén, csupán a megfelelő visszatérési kóddal tér vissza.

Lásd még

diff(I), comm(I)

Hibajelzések

A program nulla visszatérési értékkel jelzi, ha a két állomány megegyezik. Eggyel, ha különbözőek, kettővel, ha hibásan használjuk a programot, vagy nem olvasható valamelyik állomány.

M.8 comm - két állomány közös sorainak kiírása

Szintaxis

comm [-123] állomány1 állomány2

Leírás

A comm beolvassa az állomány1-et és az állomány2-t, amelyek például sorbarendezettek is lehetnek, és három oszlopos kimenetet készít. Az egyes oszlopokban megadja azon sorokat, amelyek

- csak az állomány1 nevű állományban vannak,
- csak az állomány2 állományban vannak,
- mindkét állományban szerepelnek.

Ha az állománynév helyén "-" jel áll, akkor a standard bemenetről olvas.

Opciók

Az 1, 2, vagy 3 kapcsoló a megfelelő sorszámú oszlop kiírását megtiltja.

Példák

A

```
comm -12
```

parancs csak a két állomány közös sorait fogja kiírni a

```
comm -23
```

pedig azokat, amelyek az első állományban szerepelnek, de a másodikban nem. A

comm -123

hatására pedig semmilyen eredményt sem kapunk.

Lásd még

cmp(I), diff(I), sort(1), uniq(1)

Megjegyzés

Az opciók nem a megjeleníteni kívánt oszlop sorszámát jelentik, hanem éppen azokat az oszlopokat, amelyeket nem akarunk megjeleníteni.

M.9 cp - állomány másolása

Szintaxis

cp [-ip] állomány1 állomány2
cp -rR [-ip] katalógus1 katalógus2
cp [-iprR] állomány ... katalógus

Leírás

A cp az első állományról másolatot készít a másodikba. A célállomány módja és tulajdonosa nem változik, ha már létezett ezen állomány, egyébként a forrásállomány módját és tulajdonosát használja. Ha a forrás egy szimbolikus, vagy normál link, akkor is az állomány tartalma másolódik le.

A parancs második formájában a cp rekurzívan lemásolja a katalógus1 részfát a katalógus2 alá. Ha a katalógus2 nem létezik, akkor létrehozza azt. Ha létezik, akkor a katalógus1 tartalmát egy alkatalógusként másolja be a katalógus2-be.

A parancs harmadik formájában minden megadott állomány a célkatalógusba másolódik az eredeti nevén. Ekkor a célkatalógusnak léteznie kell.

Opciók

-i Interaktív mód. Minden egyes másolás előtt a program ellenőrzi, hogy a cél-állomány létezik-e. Ha igen, akkor egy megerősítést kér, hogy azt felülírhatja-e.

-p A másolás során a forrásállomány módosítási dátuma és védelmi kódjai is lemásolódnak.

-r
-R Rekurzív másolás, azaz ha a források között van katalógus, akkor annak teljes tartalma és alkatalógusai is lemásolódnak. Ekkor a célállománynak katalógusnak kell lennie.

Lásd még

cat(1), ln(1), mv(1), tar(1)

Figyelem

A saját magába mutató rekurzív másolás végtelen ideig (amíg a diszk meg nem telik) tart. pl.

cp -r /src /src/tmp

A cp nem tartja meg a link-eket, ezért az eredeti állományok változásakor a másolatok nem változnak. A tar program alkalmas a link-ek megtartására is.

M.10 dc - asztali számológép

Szintaxis

dc [állomány]

Leírás

A dc egy nagy pontosságú aritmetikai programcsomag. Általában decimális egész számokkal dolgozik, de egyaránt megadható a bemeneti számrendszer alapja, a kimeneti számrendszer alapja, valamint a használandó törtjegyek száma. A program alapvető jellemzője, hogy ún. postfix vagy fordított lengyel módszerű (reverse Polish) logikát alkalmaz az operátorok kiértékelésére. Ha állományargumentumot is megadtunk az indításkor, akkor a bemeneti sorokat a bemenő állományból veszi, egészen az állomány végéig, majd a standard inputról folytatja.

Parancsok

szám	A szám értékét elhelyezi a verem tetejére. Számnak nevezünk egy számjegyekből álló, közvetlenül egymás után álló karaktersorozatot, amelyet egy aláhúzás "_" karakter vezethet be a negatív számok jelzésére. A szám tartalmazhat decimális pontot is.
+ - * / % ^	A verem legfelső két elemét összeadja (+), kivonja (-), összeszorozza (*), osztja (/), maradékát képzi (%) vagy hatványozza (^). Ezzel a két elemet a veremből törli is, és az eredményt a verem tetejére helyezi. Ha a hatványkitevő törtrészt is tartalmaz, akkor a törtrészt elhagyja.
sx	A verem legfelső elemét kiveszi, és elhelyezi az x nevű regiszterben, - ahol x tetszőleges karakter lehet.
Sx	Ugyanaz, mint az sx, de x-et egy független veremnek tekinti, azaz a főveremből kivett értéket az x verem tetejére helyezi.
lx	Az x regiszter értékét beleteszi a verembe. A regiszter nem változik. Minden regiszter kezdőértéke 0.
Lx	Ugyanaz, mint az lx, de x-et független veremnek tekinti, és legfelső értékét elhelyezi a főveremben.
d	A verem legfelső elemét megduplázza.
p	A verem legfelső elemének értékét kiírja a verem változatlanul hagyása mellett.
f	A veremben tárolt összes értéket kiírja.
q	A program leállása. Ha stringet hajt végre a program, akkor a rekurziós szint kettővel csökken.
Q	A verem legfelső elemét kiveszi, és a string rekurziós szint ezen értékkel csökken.
x	A verem legfelső elemét stringnek tekinti, és végrehajtja ezen karaktersort, mintha ez a dc inputja lenne.
[...]	A zárójelezett ASCII karaktersort beteszi a verem tetejére.
<x >x =x	A verem két legfelső elemét kiveszi, és összehasonlítja. Az x regiszter tartalmát akkor hajtja végre, ha az összehasonlítás eredménye igaz.
v	A verem legfelső elemét a négyzetgyökével helyettesíti. Az argumentumban (veremben levő számban) több tizedes jegy van, mint az

aktuális pontosság, akkor ezt tekinti pontossági faktornak, és nem a beállított pontosságot.

!	A sor további részét UNIX parancsként értelmezi.
c	A verem minden elemét kiveszi.
i	A verem legfelső elemét kiveszi, és ezt az értéket használja a bemenetről olvasott számok számrendszerének alapjául.
I	Az aktuális bemeneti számrendszer alapját a verembe helyezi.
o	A verem legfelső elemét a továbbiakban kiírandó számok számrendszerének alapjául veszi.
O	Az aktuális kimeneti számrendszer alapját a verembe helyezi.
k	A verem legfelső elemét kiveszi, és ezt az értéket tekinti a további pontossági faktornak, amely a figyelembe veendő törtjegyek számát jelenti a szorzás, osztás és hatványozás műveleteknél.
K	Az aktuális pontossági értéket a verem tetejére teszi.
z	A verem aktuális mélységét a verembe teszi.
Z	A verem tetején levő számot annak kiírt hosszával helyettesíti.
?	egy sort a standard bemenetről (terminálról) beolvas, és azt végrehajtja.

Példák

A következő példa hatására a program n! első tíz elemét írja ki:

```
[la1+dsa*pla10>y]
sy
0sa1
lyx
```

Lásd még

bc(I)

Megjegyzés

A bc program egy preprocesszor a dc-hez, amely lehetővé teszi a dc használatát infix operátorkiértékelési módszerrel és C-szerű strukturált vezérlő utasításokkal.

Hibajelzések

(x)?	ha az "x" karaktert nem ismeri fel,
(x)?	ha kevés a verem elemeinek száma egy parancs végrehajtásához,
Out of space	ha a szabad helyek listája túlcsordul (túl sok számjegy),
Out of headers	ha túl sok számot kell tárolnia,
Out of pushdown	ha túl sok elem van a veremben,
Nesting Depth	ha túl sok rekurziós szintet akarunk megvalósítani.

M.11 dd - állomány konvertálása és másolása

Szintaxis

dd [opció=érték] ...

Leírás

A dd lemásolja a megadott input állományt a megadott output állományba, miközben elvégzi az előírt konverziókat. Alapértelmezés szerint a standard bemenetet és a standard kimenetet használja. A bemeneti és a kimeneti blokkhosszt

külön meg lehet adni, elősegítve a közvetlen (raw) fizikai B/K-t. A következő opciók használhatók:

if=név	Név lesz bemeneti állomány neve. A standard bemenet az alapértelmezés.
of=név	Név lesz a kimeneti állomány neve. A standard kimenet az alapértelmezés.
ibs=N	A bemeneti blokkhossz N lesz (alapértelmezés: 512)
obs=N	A kimeneti blokkhossz N lesz (alapértelmezés: 512)
bs=N	Mind a bemeneti, mind a kimeneti blokkhossz egyaránt N lesz, egyúttal hatálytalanítja az ibs és az obs specifikációkat. Így, ha konverziókra nincs szükség, hatékonyabban lehet használni a programot raw eszközök másolására.
cbs=N	A konverziós puffer hossza N lesz.
skip=N	A másolás megkezdése előtt N bemeneti rekordot átlép.
count=N	Csak N bemeneti rekordot másol át.
conv=	
ascii	EBCDIC kódból ASCII kódba konvertál.
ebcdic	ASCII kódból EBCDIC kódba konvertál.
block	Változó hosszúságú rekordokat fix hosszúságúra konvertálja.
unblock	Fix Hosszúságú rekordokat változó hosszúságúra konvertálja.
lcase	A betűket mind kisbetűvé konvertálja.
ucase	A betűket mind nagybetűvé konvertálja.
swab	Felcserél minden byte-párt (big endian < - > litle endian).
noerror	Nem áll meg B/K hiba esetén.
sync	Minden bejövő rekord IBS szélességű lesz.
...,...	Különböző, egymástól vesszővel elválasztott, fentebb felsorolt konverziók.

Ahol szélességeket kell megadni, mindenütt byte-okban kell érteni. A számok mögé "k", "b" vagy "w" betűk írhatók. Hatásukra 1024-gyel, 512-vel vagy 2-vel megszorozza a számot (kilo=1024; blokk=512; word=2). Ugyanígy egy számpárt el lehet választani "x" karakterrel is, a szorzás kijelölésére.
A cbs opciót csak akkor kell megadni, ha vagy ASCII, vagy EBCDIC konverziót is kértünk. Az előbbi esetben a program cbs karaktert helyez el a konverziós pufferben, konvertálja ASCII-vá, levágja a puffer végéről a szóközöket, majd egy újsor jelet elhelyezve viszi ki a kimenetre az így kapott karaktersort. a második esetben ASCII karaktereket (újsor jelig) olvas be a konverziós pufferbe, konvertálja azokat EBCDIC-ké, majd a konverziós puffer végéig szóközöket helyez el.
Ha az átvitel véget ért, a dd beszámol a beolvasott és a kiírt rekordok számáról.

Példák

A következő példa segítségével egy 80 byte-ős EBCDIC kártyaképet lehet mágnesszalagról beolvasni, és az "x" nevű ASCII állományba konvertálni:

```
dd if=/dev/rmt0 of=x ibs=800 cbs=80  conv=ascii,lcase
```

A dd különösen alkalmas közvetlen (raw) fizikai B/K megszervezésére, mert megengedi különböző rekordhosszok alkalmazását a bemeneti és a kimeneti egységen.

Lásd még

cp(1)

Hibák

Az ASCII/EBCDIC konverziós táblák a standard CACM 1968. nov. 256 karaktert tartalmazzák. Ez nem alkalmazkodik teljes egészében a valósághoz. Az újsor karakterek beszúrása csak ASCII-ba konvertáláskor oldható meg. A sorok kitöltése sorvégi szóközökkel pedig csak EBCDIC-be konvertáláskor. Ezek különálló opciók is lehetnének.

M.12 diff - szöveges állományok soronkénti összehasonlítása

Szintaxis

diff [-l] [-r] [-s] [-cefhn] [biwt] katalógus1 katalógus2
diff [-cefhn] [-biwt] állomány1 állomány2
diff [-Dstring] [-biw] állomány1 állomány2

Leírás

A diff program normál szöveges állományok összehasonlításakor kiírja, hogy mely sorokat kell kicserélni az állományokban ahhoz, hogy a két állomány egyforma legyen. A diff megpróbálja úgy megadni az eltéréseket, hogy az a legkevesebb legyen. Ha egyik paramétere sem katalógus, akkor az egyik állománynév helyett használható a "-" karakter, ami a standard bemenetet jelenti. Ha ez egyik paraméter katalógus, a másik nem, akkor a katalógusban a megadott nevű állományt keresi meg a program. Ha mind a két paraméter katalógus, akkor a diff először névsorba rendezi a katalógusokban található állományneveket, majd azokra a szöveges állományokra, amelyek különböznek, a normál módon lefut a program. Azoknál az állományoknál, amelyek nem szövegesek és különböznek, vagy csak az egyik katalógusban szerepelnek, egyszerűen csak az állomány nevét írja ki. A szöveges állományok összehasonlításánál kapott eredmény formátuma normál esetben a következő:

n1 a n3,n4
n1,n2 d n3
n1,n2 c n3,n4

Ahol n1, n2, n3, n4 sorszámokat jelöl, az a, b, c pedig az ed editor megfelelő parancsai. Ezek a sorok az ed editor parancsaihoz hasonlóak, amelyek az első állományt a második állománnyá alakítják. A parancskódok előtti sorszámok (n1, n2) az első állományra, az azt követő sorszámok (n3, n4) pedig a második állományra vonatkoznak. Ahol a két sorszám azonos (n1 = n2, vagy n3 = n4), ott az ed editor szintaktikájának megfelelően csak az egyik szám szerepel. A sorszámpárokat visszafelé olvasva, és az a parancsot d-re cserélve előállítható az a parancssorozat, amely a második állományól állítja elő az elsőt.

A fenti formátumú sorokat azok a sorok követik, amelyeket az adott parancs érint. Minden ilyen sor előtt egy "<", vagy ">" jel áll. A "<" jel azokat a sorokat jelöli, amelyek az első állomány tartalmát változtatják meg, a ">" jel pedig azokat, amelyek a második állományt.

A -e kapcsoló hatására a generált eredményt közvetlenül fel lehet használni az ed editorral, így egy állomány különböző változatait úgy is tárolhatjuk, hogy eltesszük

az eredeti állományt, és minden változathoz eltesszük az előző változat, és az aktuális állapot összehasonlításaként kapott ed script-et. Ilyen esetben egy adott változat előállítása abból áll, hogy az ős állományon sorra végrehajtjuk a javításokat. Ennek elvégzéséhez nyújt segítséget a következő shell program, amelynek első paramétere az ős állomány, a további paraméterek pedig a javító script-ek, amit a "diff -e" adott. A megfelelő változat a standard kimeneten keletkezik.

```
(shift; cat $*; echo '1,$p') | ed - $1
```

Opciók

-b Figyelmen kívül hagy minden bevezető SPACE és TAB karaktert.

-i A kis- és nagybetűket azonosnak tekinti.

-t A tabulátor karaktereket szóközökkel helyettesíti az eredményben.

-w Figyelmen kívül hagy minden SPACE és TAB karaktert.

-c[#] Az eredményben az eltérések környezetéből # sort ír ki. Ekkor az eredmény tartalmazza az állományok nevét és módosítási idejüket, valamint az eltéréseket "*" karakterek jelzik. Azokat a sorokat, amelyeket az első állományból kell törölni, "-" jel, amelyeket a másodikba kell beszúrni "+" jel, amelyek pedig csupán megváltoztak "!" jel előz meg. A változatlan sorok előtt nem áll semmi.

-e Az ed editor a, c, d, parancsaiból álló sorozatot generál eredményként, amely közvetlenül alkalmas az első állományból a második előállítására.

-f A -e kapcsolóhoz hasonló parancssorozatot állít elő, de nem használható az ed editorhoz.

-n A -e kapcsolóhoz hasonló parancssorozatot állít elő, de ellenkező sorrendben, és megszámlálja a módosított, törölt valamint beszúrt sorokat is.

-h Csak a rövid, és jól elhatárolt változásokat ismeri fel, de gyorsabb, és tetszőleges hosszú állományokra is működik.

-Dstring
A két állományból egy közös állományt állít elő, amelyből a C preprocesszor segítségével előállítható mindkét állomány. Ha a preprocesszor számára a string azonosító definiálatlan, akkor az első állomány, ha a definiált, akkor a második generálható.

-l Hosszú eredményformátum. Katalógusok tartalmának összehasonlítása esetén minden olyan eredmény, amely szöveges állományok összehasonlításaként keletkezett, a pr program segítségével lapokra tördelve jelenik meg.

-r Rekurzív működési mód. A közös katalógusokban rekurzívan meghívja a program önmagát.

-Sname
A katalógusok összehasonlítását a nevek rendezése után a name nevű állománnyal kezdi.

Lásd még

cc(1), cmp(1), comm(1), cpp(1), diff3(1), ed(1), pr(1)

Hibajelzések

A program nulla visszatérési értékkel jelzi, ha a két állomány megegyezik. Eggyel, ha különbözőek, kettővel, ha hibásan használjuk a programot, vagy nem olvasható valamelyik állomány.

M.13 dirname - katalógusnév kiírása a névből

Szintaxis

dirname név

Leírás

A dirname csak az állománynévhez vezető útnevet írja ki a névből. Azaz az utolsó '/' jellel határolt rész kivételével mindent kiír a megadott névből. Ha a név nem tartalmaz '/' jelet úgy csak egy '.' karaktert ír ki.

M.14 du - diszkhasználat összegzése

BSD szintaxis

du [-s] [-a] [állománynév]

System V szintaxis

du [-s] [-a] [-r] [állománynév]

Leírás

A du program megadja, hogy a megadott állományok, illetve a katalógusok (ez utóbbiak rekurzív módon minden általuk tartalmazott újabb bejegyzés tartalmával együtt) mennyi helyet foglalnak el a diszken. A többször bejegyzett, azonos i-node számú állományokat csak egyszer számítja bele a hosszba. Ha az állománynév argumentum hiányzik, akkor a munkakatalógust használja helyette.

System V leírás

A system V rendszerekben a du nem Kb egységben, hanem fél Kb-os egységben adja meg a foglalt helyet.

Opciók

-s Csak a teljes összeget adja meg.

-a Minden állomány méretét külön-külön megadja.

Ha nem adtunk meg kapcsolót, akkor csak a katalógusok, illetve az ezek által tartalmazott újabb katalógusok hosszát írja ki.

System V opciók

-r System V alatt normál esetben a du nem ad hibajelzést, ha egy katalógust nem tud megnyitni. A -r kapcsoló hatására ad.

Hibák

Ha nem katalógusokat adunk meg argumentumként, és a -a kapcsolót nem használjuk, akkor nem listázza ki őket.

M.15 ed - szövegszerkesztő

Szintaxis

ed [-] [sx] [-p string] [állomány]

Leírás

Az ed a UNIX standard szövegszerkesztője. Ha az állomány paramétert megadtuk, akkor a program indulásakor egy "e állomány" parancsot hajt végre, majd újabb parancsra várakozik. A program működésének részletes leírását az editorokkal foglalkozó E függelék tartalmazza.

Opciók

-

-s Az e, r, w, parancsok végrehajtását kísérő üzenetek nem íródnak ki. Nem íródnak ki továbbá az edit puffer felülírására figyelmeztető üzenetek sem. Akkor használatos ezen opció, ha nem interaktív módon használjuk a programot.

-x Titkosított állomány javítása.

-p string
Amikor parancsra vár, a string-et írja ki, mint prompt stringet.

Lásd még

crypt(1), ex(1), grep(1), sed(1), sh(1), vi(1)

M.16 expr - kifejezés kiértékelés

Szintaxis

expr kifejezés

Leírás

Az expr kiértékeli az adott kifejezést, majd az eredményt a standard outputra írja. A kifejezés minden tagját külön paraméterként kell átadni. Az operandusok egész számok, vagy stringek lehetnek. Egyes, a shell által is értelmezett karaktereket idézőjelek közé zárva kell átadni.

A következő operátorok használhatók (növekvő precedencia szerint):

|	logikai vagy operátor
&	logikai és operátor
< <= = == != >= >	relációk
+ -	összeadás, kivonás
*** / %**	szorzás, osztás, maradékképzés
:	mintaillesztés. Pl. par1 : par2, ahol par2 egy reguláris kifejezés, melynek az elején egy implicit '^' van. A ':' operátor a par1-re illeszti a reguláris kifejezést. Ha a kifejezés tartalmaz '\(' és '\)' alakot, akkor az expr a '\1' -nek megfelelő stringet, egyébként az illeszkedő karakterek számát írja ki a standard output-ra.
( )	zárójelek

Példák

Az **a** shell változó növelése:

```
a=`expr $a + 1`
```

A **j** shell változó kétszeresének kiírása:

```
echo `expr $j '*' 2`
```

Az **a** shell változón végrehajt egy basename-nek megfelelő transzformációt:

```
a=`expr $a : '.*/\(.*\)' '|' $a`
```

Hibajelzések

A program a következő visszatérési értékeket adja:

0 ha a kifejezés értéke nem nulla,
1 ha a kifejezés értéke null vagy nulla
2 érvénytelen kifejezés esetén

M.17 file - állomány típusának meghatározása

Szintaxis

file [-f ffile] [-cL] [-m mfile] állomány ...

Leírás

.A file program minden egyes argumentumán végrehajt egy tesztsorozatot, hogy meghatározza a típusát. Ha az argumentum szövegállomány, akkor a file program megvizsgálja első 512 byte-ját, és megpróbálja meghatározni, hogy milyen programnyelven van írva, vagy milyen program bemeneti állománya lehet. A program a /etc/magic állományt használja a mágikus számmal kezdődő állományok típusának meghatározására. Így bármelyik állomány tartalmazhat egy numerikus vagy karakteres konstanst, ami a file típusát azonosítja.

Opciók

-c Ellenőrzi a mágikus számokat tartalmazó állomány szintaktikai felépítését. Az ellenőrzés után a program megáll.

-f ffile
A típus meghatározásra előírt állományok neveit az ffile állományból veszi.

-L Ha az állomány szimbolikus link, akkor nem a hivatkozás típusát határozza meg, hanem az állományét.

-m mfile
Az mfile állomány használja a /etc/magic helyett.

Hibák

Gyakran a shell parancsállományokra azt mondja, hogy C program.

M.18 find - állományok keresése nevük vagy tulajdonságaik alapján

Szintaxis

find útnév_lista kifejezés
find minta

Leírás

A find program rekurzív módon keres az útnév_lista minden elemétől kezdődően. Az útnév_lista minden katalógusától kezdődően a teljes fát bejárja, és megkeresi azon állományokat, amelyek megfelelnek a megadott logikai kifejezésnek, amelyet az alább definiált elsődleges kifejezések alkothatnak. A find nem követi a szimbolikus link-eket. A megadott feltételt magára a link-re alkalmazza, mintha az normál file lenne.

A használat második formájában a find program egy előre felépített adatbázisból keres a megadott minta alapján. Az adatbázist a rendszergazdának kell felépíteni

pl. hetenként, és ez tartalmazza az összes publikusan elérhető állomány nevét. A mintában a shell-nél megismert állományhelyettesítő karakterek (*, ? [ ]) használhatók.

a leírásban N egy decimális egész számot jelent, ahol +N jelentése nagyobb, mint N, -N jelentése kisebb, mint N, N jelentése pedig pontosan egyenlő N.

Kifejezések:

A paraméterként megadott kifejezés a következő elsődleges kifejezésekből állhat:

-name file	A kifejezés igaz, ha a file argumentum megegyezik az aktuális állománynévvel. A shell által megértett állománynév-helyettesítő szintaxis használható, de idézőjelek közé kell zárni ezeket, hogy ne a shell értelmezze.
-perm onum	Igaz, ha az aktuális állomány védelmi kódja pontosan megegyezik az onum oktális számmal (lásd chmod(I)). Ha az onum számot mínusz jel előzi meg, akkor több bit adható meg (017777). Ezen bitek az állomány módját és típusát jellemzik. Ekkor az onum a következőképpen értelmeződik: (flag & onum) == onum.
-type c	a kifejezés igaz, ha az aktuális bejegyzés típusa c, ahol c jelentése: b - blokk speciális állomány, c - karakter speciális állomány, d - katalógus, f - normál állomány, p - named pipe, l - szimbolikus link, s - socket.
-links N	Igaz, ha az aktuális állományra N link van.
-user uname	Igaz, ha az állomány tulajdonosa az uname nevű felhasználó.
-nouser	Igaz, ha az állomány tulajdonosa nincs a /etc/passwd állományban regisztrálva.
-group gname	Igaz, ha az állomány tulajdonosa a gname csoportba tartozik.
-nogroup	Igaz, ha az állomány tulajdonosa nincs a /etc/group állományban regisztrálva.
-size N	Igaz, ha az állomány N blokk hosszúságú (512 byte van egy blokkban).
-inum N	Igaz, ha az állomány i-node száma N.
-atime N	Igaz, ha az állomány hozzáférési ideje N (napban megadva).
-mtime N	Igaz, ha az állomány módosítási ideje N (napban megadva).
-exec program	Igaz, ha a végrehajtandó program 0 visszatérési értékkel tér vissza (a legtöbb parancs ilyen). A parancs végét egy pontosvessző ";" jelzi, amit a shell miatt idézőjelek közé kell zárni. Ha a parancsnak üres kapcsos zárójelpárt "{}" adunk meg argumentumnak, akkor ennek helyére az aktuális állománynevet fogja elhelyezni.
-ok program	a parancs megegyezik az -exec paranccsal, de az előállított parancssort a find kinyomtatja (előtte egy kérdőjellel), és csak akkor hajtja végre, ha a felhasználó válasza "y".

-print	a kifejezés mindig igaz; hatására az aktuális állomány teljes nevét kiírja a standard output-ra.
-ls	Mindig igaz. Az aktuális állománynév és a hozzá tartozó adatok olyan formában íródnak ki, mintha az ls -gilds parancs állította volna elő.
-newer file	Igaz, ha az aktuális állomány újabb a file argumentumban megadott állománytól.
-cpio file	Az aktuális állományt a file nevű állományba írja cpio formátumban.
-xdev	Mindig igaz, hatására a find program nem megy át másik kötetre (mount-olt eszközre).

Operátorok:

Az elsődleges kifejezések a következő operátorokkal kombinálhatók (az itt megadott sorrendben precedenciájuk csökken):

(kifejezés)	Zárójelezés segítségével lehet a kifejezéseket csoportosítani. A zárójeleket idézőjelbe kell zárni, hogy ne a shell értelmezze.
!	NEM (logikai tagadás).
-a	ÉS (logikai és) kapcsolat; a második operandust csak akkor értékeli ki, ha az első értéke igaz. Ha két kifejezés között nem áll semmilyen operátor, akkor az az ÉS kapcsolatnak felel meg.
-o	VAGY (logikai vagy) kapcsolat; a második operandust csak akkor értékeli ki, ha az első értéke hamis.

Példák

Példaként megadunk egy parancssort, amelynek segítségével az "a.out" és a "*.o" nevű állományokat lehet törölni, ha azokat egy hétnél régebben használták.

```
find / "(" -name a.out -o -name "*.o" ")" -a \
-atime +7 -a -exec rm {} ";"
```

Lásd még

chmod(1), cpio(1), ln(1), ls(1), sh(1), test(1)

M.19 grep, egrep, fgrep - állományban minta szerinti keresés

Szintaxis

grep [opció] ... kifejezés [állomány] ...
egrep [opció] ... [kifejezés] [állomány] ...
fgrep [opció] ... [stringek] [állomány] ...

Leírás

A grep parancsok a megadott bemeneti állományokban (a standard input az alapértelmezés szerinti bemenet) olyan sorokat keresnek, amelyek a reguláris kifejezéssel megadott mintára illeszkednek. A megtalált sorokat általában a standard kimenetre írják ki. A grep mintaillesztője az ed editor mintaillesztőjének egyszerűsített változata. Az egrep mintaillesztője az összes reguláris kifejezést ki tudja értékelni,

de ehhez több memóriára van szükség. Az fgrep mintaillesztője fix stringeket használ, ezért ez a leggyorsabb és legegyszerűbb. Minden esetben, ha egynél több állománynevet adtunk meg, a megtalált sorok elé kiírja azt is, hogy melyik állományban vannak.

Opciók

-h Nem írja ki az állományok nevét.
-i A kis- és nagybetűket azonosnak tekinti.
-l Csak az állományok nevét írja ki.
-n Minden kiírt sor előtt a sor sorszáma is szerepel.
-s Nem ír ki semmit, csak a visszatérési értékben ad információt.
-v a nem illeszkedő sorokat írja ki.
-w A kifejezést mint szót keresi. (csak a grep)
-x Csak azokat a sorokat írja ki, amelyek teljes egészében illeszkedtek.
-e kifejezés
Ugyanaz, mint a kifejezés argumentum, de akkor használatos, ha a kifejezés "-" jellel kezdődik.
-f file
A reguláris kifejezéseket (egrep) vagy a stringeket a file nevű állományból veszi. Az állomány minden sora tartalmazhat egy kifejezést vagy stringet.

Megjegyzés

A $, *, [,], ^, |, (,), és \\ karaktereket a shell maga is értelmezi, így általában ezeket idézőjelek között kell megadni.

Lásd még

ed(I), sh(I)

Hibák

a sorok maximális hossza 1024 karakter lehet; a hosszabb sorokat csonkítják a programok.

M.20 head - első néhány sor kiíratása

Szintaxis

head [-szám] [állomány]...

Leírás

A head program a megadott állomány vagy a standard bemenet első néhány sorát írja ki a standard kimenetre. Ha -szám argumentum meg van adva, akkor ennek megfelelő számú sort ír ki, egyébként 10 sort.

Lásd még

tail(1)

M.21 join - relációs adatbázis-illesztő

Szintaxis

join [opciók] file1 file2

Leírás

A join a file1 és a file2 állományok megfelelő sorait összeilleszti, és kiírja az eredményt a standard kimenetre. Ha a file1 helyén "-" áll, akkor a standard bemenetről veszi a sorokat.

A két állománynak a kulcsmezők szerint növekvő sorrendben rendezettnek kell lennie. A kulcsmező alapértelmezés szerint az első mező a sorban.

Minden olyan mezőhöz, amelyek megegyeznek a két állományban, egy eredménysor tartozik. Ez a sor először a közös mezőt, ezután a file1-ból a mező nélküli részt, majd a file2-ből a mező nélküli maradéksort tartalmazza.

A mezők rendszerint szóközökkel, tabulátorokkal vagy újsorjellel vannak elválasztva. Ebben az esetben a többszörös szeparátorok egynek számítanak, és a bevezető szeparátorok figyelmen kívül maradnak.

Opciók

-an A normál eredményen kívül, azokat a sorokat is kiírja a filen-ből, amelyekhez nem sikerült megfelelő mezőt találni. Az n értéke 1 vagy 2 lehet.

-e s Az üres eredménymezőket az s stringgel helyettesíti.

-jn m Az m. mezőt használja az n. file-ból. Ha n hiányzik, akkor mind a két file-ban az m. mezőt használja. n ill. m értéke 1 vagy 2 lehet.

-o list Az eredménysorok a listában megadott mezőket tartalmazzák. A lista minden eleme m.n formátumú, ahol m a mező, n a file sorszáma. Az egyes elemeket szóközzel vagy vesszővel kell elválasztani egymástól.

-tc Mezőszeparátorként a c karaktert használja. (c-nek Minden előfordulása számít.)

Lásd még

sort(1), comm(1), awk(1)

M.22 kill - folyamat megszüntetése

Szintaxis

kill [-szignál] folyamatszám ...
kill -l

Leírás

A kill parancs segítségével a megadott számú (pid) folyamatoknak egy ún. szignált lehet küldeni. Ha nincs megadva a szignál argumentum, akkor a kill automatikusan SIGTERM (terminate, 15) szignált fogja elküldeni. A szignálokra számukkal vagy nevükkel lehet hivatkozni. A -l kapcsoló megadásakor a program kiírja az összes lehetséges szignál nevét. A legfontosabbak ezek közül:

SIGHUP(1) - hangup
SIGINT(2) - interrupt karakter (ctrl-c)
SIGQUIT(3) - quit karakter (ctrl-\)
SIGKILL(9) - kill
SIGTERM(15) - terminate

Ha a folyamat számaként a "0"-t használjuk, akkor az aktuális felhasználó minden folyamata megkapja a megadott szignált. (Csak Bourne Shell).
Szignált csak a folyamat tulajdonosa vagy a super-user küldhet.

Lásd még

csh(1), ps(1), sh(1), signal(2), init(8)

M.23 ln - normál link vagy szimbolikus link készítése

BSD szintaxis

ln [-fs] állománynév [linknév]
ln [-fs] állománynév... katalógus

System V szintaxis

ln [-fFs] állománynév [linknév]
ln [-fFs] állománynév... katalógus

Leírás

Az ln egy újabb katalógusbejegyzést készít a megadott állományra, ami katalógus is lehet. Egy állományra tetszőleges számú hivatkozás (link) lehet. Egy ilyen hivatkozással az állomány egyéb attribútumai (hossz, védelem, stb) nem változnak meg. A parancs állománynév paraméterként az eredeti állomány nevét, linknév paraméterként pedig az új nevet kell megadni. Ha ez utóbbi nincs megadva, akkor az eredeti állománynév utolsó komponensét használja a program az új bejegyzés elkészítéséhez.

A parancs második formájában az utolsó paraméter egy katalógusnév. Ekkor ebben a katalógusban keletkezik egy-egy újabb hivatkozás az összes megadott állományra.

Az egyszerű link (ez az alapértelmezés) a normál katalógusbejegyzéssel megegyezik. Ilyen linket csak létező állományra lehet készíteni, továbbá az egyszerű link nem nyúlhat át másik kötetre (mount-olt egységre).

A szimbolikus link, amit a -s kapcsolóval lehet létrehozni, egy speciális katalógusbejegyzés, ami egy másik állománynévre mutat. A szimbolikus link másik állományrendszerre és katalógusra is mutathat. Az is lehetséges, hogy a szimbolikus link olyan állományra mutasson, amely pillanatnyilag nincs az állományrendszerben. Ha egy olyan állományt törlünk, amelyre szimbolikus link van, akkor az nincs hatással a linkre, azaz a szimbolikus hivatkozás megmarad, de az állomány letörlődik. Ha egy szimbolikus link katalógusra mutat, akkor annak más "furcsa" tulajdonságai is vannak:

- Az ls parancs a link-elt katalógus tartalmát mutatja meg, az ls -l viszont csak a linket, mint bejegyzést.
- Ha egy ilyen katalógusba cd-vel bemegyünk, akkor a cd .. nem oda visz vissza, ahol a szimbolikus link volt, hanem a katalógus tényleges szülőkatalógusára.

BSD opciók

-f Engedélyezi egyszerű link készítését katalógusra is. Ezt csak a szuper-user engedélyezheti.

-s Szimbolikus linket készít.

System V opciók

Minden opció ugyanazt jelenti, és a -f kapcsoló nagybetűvel is írható.

Lásd még

rm(1), link(2), unlink(2)

Figyelmeztetés

Szimbolikus link készítésekor, ha a létrehozott link állomány nem abban a katalógusban van, ahol az eredeti, akkor az eredeti állományt teljes útnévvel célszerű megadni. Ugyanis a szimbolikus link nem más, mint egy olyan katalógusbejegyzés, amely a megadott eredeti nevet tartalmazza. Ha ez nem teljes útnév, akkor az ahhoz a katalógushoz képest relatívan értendő, ahova a link bejegyzés tartozik. Ezek alapján a

```
cd ; echo mit sutsz kis szucs >file ; mkdir dir
cd dir ; echo duma >file ; ln -s file ../alma
cd .. ; cat alma
```

utasítássorozat a "mit sustsz kis szucs" szöveget fogja eredményezni.

M.24 ls - a katalógus tartalmának kilistázása

BSD szintaxis

ls [-aAcCdfFgilLqrRstu1] [állománynév...]

System V szintaxis

ls [-abcCdfFgilLmnopqrRstux] [állománynév...]

Leírás

Az ls kilistázza az összes megadott katalógus tartalmát. Ha a paraméterként kapott állomány nem katalógus, akkor annak nevét a kapcsolókkal kiválasztott információkkal kiegészítve kiírja. Ha nem adtunk meg egy állománynév argumentumot sem, akkor az éppen aktuális munkakatalógus tartalmát listázza ki. Az ls a-lapértelmezés szerint abc sorrendben listáz. Ha több névargumentumot adunk meg, akkor a következő eljárást alkalmazza:

1. névsorba szedve kilistázza az összes állomány-argumentumot;
2. névsorba szedi a katalógus-argumentumokat;
3. egymás után kilistázza a katalógusok tartalmát, névsorba szedve tartalmukat.

Az állomány attribútumait a -l kapcsolóval ún. hosszú formátumban kapjuk meg. A védelmi módot egy 10 karakter hosszú sorozat jelzi. Ennek felépítése a következő:

drwxrwxrwx

Ahol az első karakter:

d ha a bejegyzés katalógus,

b ha a bejegyzés blokk-orientált speciális állomány,
c ha a bejegyzés karakter-orientált speciális állomány,
l ha a bejegyzés szimbolikus link,
p ha a bejegyzés ún. named pipe,
s ha a bejegyzés socket,
- ha a bejegyzés egy egyszerű állomány.

A következő 9 karakter hármas csoportokra osztható. Az első csoport a tulajdonos, a második a csoporttárs, a harmadik hármas pedig bárki más számára engedélyezett jogokat tükrözi. A hármas csoporton belül a három karakter sorrendben a következő lehet: "**r**", "**w**" és "**x**". ezek bármelyike helyén "**-**" is állhat. Jelentésük pedig rendre az olvasási (**r**ead), írási (**w**rite), végrehajtási (e**x**ecution) jogot vagy "-" esetén annak hiányát jelenti.

A tulajdonos lehetőségeit ismertető csoportban "**x**" helyén "**s**" áll, ha az állomány setuid módban van. Hasonlóképpen a csoporttárs lehetőségeit ismertető csoportban "**x**" helyén "**s**" áll, ha a bejegyzés setgid módú.

Ha az utolsó karakter (ami általában "x", vagy "-") helyén "t" áll, akkor az az állomány védelmi módjában az ún. stiky bitet (1000-es bit) jelenti. Ha ugyanekkor az állomány setuid módú vagy végrehajtható, akkor azt az "S" ill. "T" jelzi.

BSD opciók

-a Minden bejegyzést kilistáz. Ezen opció nélkül a "."-al kezdődő nevű bejegyzéseket nem listázza ki, kivéve ha a programot super-user használja.
-A Ugyanaz mint a -a ,de a ".", és a ".." állományokat mégsem írja ki.
-c Az i-node módosítási időt használja a kiíráshoz és a rendezéshez.
-C Több oszlopban írja ki a bejegyzéseket. Ez az alapértelmezés, ha az ls a terminálra dolgozik.
-d Ha a paraméter katalógus, akkor csupán a nevét, nem pedig a tartalmát listázza ki. Célszerűen a -l kapcsolóval együtt érdemes használni a katalógus módjának meghatározására.
-f Minden argumentumot katalógusként értelmez, és kilistáz minden bennük tárolt nevet. Ezen kapcsoló használata esetén nem rendezi a neveket, hanem úgy írja ki, ahogy megtalálja a katalógusban. Az opció kikapcsolja a "-l", "-t", "-s" és "-r" kapcsolókat, és bekapcsolja a "-a" kapcsolót.
-F Minden katalógust "/" jellel, a végrehajtható állományokat "*" jellel, a szimbolikus linkeket "@" jellel, a socket-eket pedig egy "=" jellel jelöl a nevük végén.
-g A tulajdonos csoportját is kiírja a hosszú listában, nemcsak a tulajdonost.
-i Minden egyes bejegyzés i-node számát is megadja a lista első oszlopában.
-l Hosszú formában listáz. Minden egyes állománynak megadja a módját, a katalógus-hivatkozások számát (link), a tulajdonos nevét, byte-okban az állomány hosszát, valamint az utolsó módosítás időpontját. Ha a bejegyzés speciális állomány, akkor a hosszmező az ún. major és minor számot tartalmazza. Ha az utolsó módosítási ideje régebbi mint 6 hónap, akkor azt "hónap nap év" alakban, egyébként "hónap nap időpont" alakban írja ki. A szimbolikus link-eknél "->" karaktersorozat után megadja az eredeti hivatkozott nevét is.

-L Ha a megadott paraméter szimbolikus link, akkor a hivatkozott állomány ill. katalógus nevét és attribútumait írja ki a link helyett.
-q Az állománynevekben levő nem látható karaktereket "?" jellel helyettesíti. Ha az ls a terminálra dolgozik, akkor ez alapértelmezés.
-r Fordított sorrendben rendez. A névsor szerinti rendezés helyett fordított abc sorrend szerint listáz, illetve ha a -t kapcsolóval együtt használjuk, akkor a legrégebbi idővel rendelkező állomány listázódik ki előbb.
-R Rekurzívan bejárja az alkatalógusokat is.
-s A bejegyzés hosszát 1 Kb-os blokkokban adja meg byte-ok helyett. Ebbe a hosszba az indirekt blokkokat is beszámítja.
-t A névsor szerinti listázás helyett időrendi sorrendben fog listázni. (A legfiatalabb idővel rendelkező állomány listázódik ki előbb.)
-u Az utolsó módosítás ideje helyett az utolsó hivatkozás (használat) ideje szerint rendez, ha "-t" kapcsolóval együtt áll. A "-l" kapcsolóval együtt pedig ezt az időt listázza, nem a módosítás idejét.
-1 Egy oszlopban listáz. Ez az alapértelmezés, ha az eredmény nem terminálra megy.

System V opciók

Minden itt fel nem sorolt kapcsoló azonos működést eredményez a BSD rendszerbelivel.

-a Ugyanaz, mint a BSD rendszerekben, de a kapcsoló nélkül a super-user számára sem írja ki a pont karakterrel kezdődő állományokat.
-g Ugyanaz mint a -l, de nem írja ki a tulajdonos azonosítóját, csak a csoportazonosítót.
-l Ugyanaz, mint a BSD rendszerekben, de a csoportazonosítót is kiírja.
-s Ugyanaz, mint a BSD rendszerekben, de 512 byte-os blokkokban értendő.
-b A nem látható karaktereket \ddd alakban írja ki.
-m Az állományok neveit egy sorba írja (amennyi elfér egy sorban), és vesszővel választja el egymástól.
-n Ugyanaz, mint a -l, de a tulajdonos- ill. csoportazonosító helyett az azonosító számokat írja ki.
-o Ugyanaz, mint a -l, de a csoportot nem írja ki.
-p Minden katalógus neve után egy "/" jelet ír.
-x Többoszlopos lista, ahol a rendezés vízszintes irányú.

M.25 man - kézikönyv lapjainak kiíratása, keresés a kézikönyvben

Szintaxis

man [-] [-M útnév] [-T makro_csomag] [[fejezet] parancs...] ...
man [-M útnév] -k kulcsszó
man [-M útnév] -f állománynév

Leírás

A man program segítségével a UNIX kézikönyv egyes fejezeteit, illetve fejezeteinek részeit lehet kiíratni a standard kimenetre. A kimenet formája képernyő-orientált, ha a kimenet a terminál.

A man parancs az első megadási formában teljes leírásokat ad. Attól függően, hogy adtunk-e meg fejezet argumentumot, csak az adott fejezetben vagy az egész kézikönyvben keresni fogja a megadott parancsok leírását. A fejezet megadható számmal és az esetlegesen azt követő betűvel vagy a következő szavak egyikével: new, local, old, public.

Ha a man program megtalálja a kérdéses leírást leformázva, akkor azt írja ki, ha nem, akkor a megfelelő formázó programokkal először leformázza. Normál esetben a more program segítségével jeleníti meg az eredményt, ha viszont a standard kimenet nem terminál, vagy a "-" kapcsolót megadtuk, akkor a cat programot használja.

Opciók

-M útnév

Az útnévben megadható azoknak a katalógusoknak a listája, amelyek a leírások alkatalógusait tartalmazzák. Az egyes útneveket pontosvesszővel kell elválasztani. Pl: /usr/man/u_man:/usr/man/a_man. A -k , és a -f opciók használatakor a -M opciót kell először megadni.

-T makro_csomag

A leírások formázásához a megadott makrócsomagot fogja használni, és nem a standard -man (troff(1), nroff(1)) makrókat.

-k kulcsszó

A man program kiírja az összes olyan leírás egysoros összefoglalóját a whatis adatbázisból (ami a tartalomjegyzéknek felel meg), amelyben a megadott kulcsszó szerepel. A whatis adatbázist a rendszergazdának kell létrehoznia a catman(8) paranccsal.

-f állománynév

A man megpróbálja kiírni a megadott állománynevekhez tartozó egysoros összefoglalót, mégpedig úgy, hogy a megadott állománynévből eltávolítja az útnevet és a maradékot, vagy annak a kiterjesztés nélküli részét (basename) a leírások összefoglalóiban keresi.

Környezeti változók

MANPATH	Ha létezik, akkor az ebben megadott út alapján keresi a leírásokat tartalmazó katalógusokat. A -M kapcsoló felüldefiniálja a MANPATH változó hatását.
PAGER	A változóban megadott programot használja a megjelenítéshez a more helyett.
TCAT	A troff programmal formázott leírások megjelenítő programja adható meg ebben a változóban.

Lásd még

apropos(1), cat(1), col(1), eqn(1), lpr(1), more(1), nroff(1), refer(1), tbl(1), troff(1), whatis(1), eqnchar(7), man(7), catman(8)

M.26 mkdir - új katalógus létrehozása

Szintaxis

mkdir [-p] katalógusnév...

Leírás

Az mkdir parancs létrehozza az argumentumaként megadott katalógusokat. A -p kapcsoló engedélyezi, hogy a megadott katalógushoz vezető út hiányzó katalógusai is keletkezzenek. A létrejövő katalógusok védelmi bitjeit az aktuális umask(2) beállítás határozza meg. Természetesen a program csak akkor hozza létre a megadott katalógusokat, ha a megfelelő szülő katalógusba van írási joga az adott felhasználónak. A katalógusok tulajdonosa az a személy lesz, aki a parancsot kiadta. A program az egyes katalógusok standard "." és ".." bejegyzéseit automatikusan elkészíti.

Lásd még

chmod(1), rm(1), rmdir(1), csh(1), umask(2)

M.27 mv - állomány átnevezése vagy átmásolása

Szintaxis

mv [-] [-fi] állomány1 állomány2
mv [-] [-fi] katalógus1 katalógus2
mv [-] [-fi] állomány ... katalógus

Leírás

Az mv parancs állományokat mozgat az állományrendszerben, aminek egy mellékhatása az, hogy átnevezi az állományt.

A parancs az első megadási formában átnevezi az állomány1 nevet állomány2-re. Ha az állomány2 már létezik, akkor az előbb letörli. Ha az állomány2 védelmi bitjei olyanok, hogy az nem írható, akkor kiírja a standard kimenetre a védelmi biteket oktálisan (ld. chmod(2)), és sort beolvas a standard bementről. Ha a válasz sor y karakterrel kezdődik, akkor letörli az állomány2-t, egyébként nem történik meg az átnevezés.

A második megadási formában az mv parancs a katalógus1 nevet katalógus2 névre változtatja. Ha a katalógus2 már létezik, akkor a harmadik megadási formánál leírtak érvényesülnek.

A harmadik megadási formában az mv átmozgat egy vagy több állományt (ami katalógus is lehet) az utolsóként megadott katalógusba. Ebben az eredeti névvel fognak szerepelni a megadott állományok.

Az mv visszautasítja, ha egy állományt vagy katalógust saját magára szeretnénk átnevezni.

Opciók

- Minden további paramétert állománynévnek értelmez. Így lehetséges "-" jellel kezdődő nevet megadni első paraméterként.

-f Üzenet kiírása nélkül felülírja a meglevő állományt, azaz nem vár megerősítő választ, ha az állomány nem írható, hanem végrehajtja az átnevezést.

-i Interaktív mód. Az mv kiírja az állomány nevét és egy kérdőjelet, ha az átnevezés során már meglevő állományt írna felül. Ezután egy sort olvas a

standard bemenetről. Ha ez a sor y karakterrel kezdődik, akkor letörli az állományt, egyébként nem történik meg az átnevezés.

Hibák

Ha a forrás és a cél nem ugyanazon az köteten (mount-olt eszközön) helyezkedne el, vagyis ha az egyszerű átnevezés nem oldható meg, akkor a program lemásolja az állományt, majd törli az eredetit. Ekkor viszont a tulajdonos és a módosítási idő a cp(1) parancsnak megfelelően megváltozik, valamint a linkek is elvesznek.

Az mv nem mozgatja át katalógust az egyik kötetről a másikra. Erre a feladatra a cp(1) parancs alkalmas.

M.28 od - oktális, decimális, haxadecimális és ASCII dump

BSD szintaxis

od [-format] [állomány] [[+]offset[.][b] [címke]]

System V szintaxis

od [-formát] [állomány] [[+]offset[.][b] [címke]]

BSD leírás

Az od a megadott állományt, vagy annak hiányában a standard bemenetet az első argumentum által kiválasztott egy vagy több dump formátumban kiírja. Ha az első argumentum hiányzik, akkor az alapértelmezés az oktális (-o) formátum. A kiírás az állomány elején kezdődik, ha nincs offset argumentum megadva, és az állomány végéig tart. Ha van offset argumentum, akkor az offset az állománynak azt a pontját adja meg, ahol a kiírásnak kezdődnie kell. Ezt az argumentumot a program normál esetben oktális byte-okként értelmezi. Ettől eltérő számrendszer is megadható: Ha az offset után "." szerepel, akkor az offset decimálisan értelmeződik, ha pedig "x" vagy "0x" van az offset előtt, akkor azt hexadecimálisan értelmezi. Az offset mögött "b" jelzi, ha 512 byte-os blokkokban adtuk meg az offset értéket. Ha az állomány argumentum hiányzik, az offset argumentum előtt "+"-nak kell állnia.

Ha van offset paraméter, akkor az egyes sorok elején kiírt cím az offset számrendszerével azonos számrendszerben íródik ki. Egyébként a kiírt cím oktális.

A címke egy látszólagos címként értelmeződik, és a cím után kerül kiírásra kerekzárójelek között. A címke megadása azonos az offset megadásával.

Formátumkódok

- a Az állomány byte-jait karaktereknek értelmezi, és azok ASCII nevét írja ki. Ha a formátumkódban a "p" karaktert is megadjuk, akkor a páros, ha viszont a "P" karakter, akkor a páratlan paritású byte-okat aláhúzva írja ki. Egyébként a paritásbitet figyelmen kívül hagyja.
- b A byte-okat oktálisan értelmezi.
- c A byte-okat ASCII karaktereknek értelmezi. A nem látható karaktereket 3 jegyű oktális kódjukkal írja ki \nnn formában. Néhány vezérlő karaktert rövidítve ír ki: NUL=\0, BACKSPACE=\b, FORMFEED=\f, NEWLINE=\n, RETURN=\r, TAB=\t.
- d Az állomány byte-jait előjeletlen egész szavaknak (short word) értelmezi, és ezek értékét írja ki decimálisan.

f A szavakat (long word) lebegőpontos decimális számoknak értelmezi.
h A szavakat (short word) hexadecimális számoknak értelmezi.
i A szavakat (short word) előjeles decimális számoknak értelmezi.
l A szavakat (long word) előjeles decimális számoknak értelmezi.
o A szavakat (short word) oktálisan értelmezi.
s[n] Azokat a látható ASCII karaktersorozatokat írja ki, amelyek NUL byte-al végződnek, és hosszabbak mint n byte. A hossz alapértelmezés szerint 3 byte.
v Minden adatot kiír. Alapértelmezésben azokat a sorokat, amelyek az előző kiírt sorral azonos tartalommal jelennének meg, csak egy "*" jel helyettesíti a képernyő bal szélén.
w[n] Megadja, hogy a képernyő egy sorába hány bemeneti byte kerüljön. Ha nincs megadva a w, akkor ez 16. Alapértelmezésben n=32.
x A szavakat (short word) hexadecimálisan értelmezi.

A számformátumok értelmezésénél a nagybetű hosszú (long) vagy dupla pontosságot ír elő. (BDFHILOX)

System V leírás

A program működése egy helyen tér el a BSD rendszertől: Az "s" formátum előjeles decimális formátumnak értelmeződik, és a "S" formátum keresi a karaktersorozatokat.

Lásd még

adb(1), dbx(1), dbtool(1)

Hibák

Az állományparaméter nem kezdődhet "+" karakterrel. Hexadecimális offset nem adható blokk offset-nek. Csak egy bemeneti állományt tud kezelni.

M.29 passwd, chfn, chsh - jelszó információ megváltoztatása

Szintaxis

passwd [-f] [-s] név

Leírás

A passwd parancs segítségével a név hívónevű felhasználó jelszava, bejelentkező shell-je (-s opció), vagy az ún. GECOS információs mezője (-f opció) változtatható meg. A chsh azonos a "passwd -s", a chfn pedig azonos a "passwd -f" paranccsal.

A jelszó megváltoztatásához a program megkérdezi a régi, majd az új jelszót. Mivel egyik jelszó sem látszik beíráskor, ezért az új jelszót a program még egyszer megkérdezi, mielőtt elfogadja. Így a hibás gépelésből adódó "jelszóvesztés" elkerülhető.

A chsh parancs kiírja az éppen aktuális bejelentkező shell nevét, majd kéri az új shell program nevét. Új shell-ként csak olyan program adható meg, amely a /etc/shells állományban fel van sorolva. Ha ilyen nincs, akkor csak a /bin/sh ill. a /bin/csh adható meg. (Természetesen a super-user ettől eltérhet.)

A chfn parancs segítségével megváltoztatható password állományban találgató ún. GCOS információs mező tartalma. A program ezt a mezőt több almezőre osztja, és így írja ki, ill. kéri be az új adatokat. Ezek a következők: a felhasználó valódi neve, munkahelyi szobaszáma, munkahelyi telefonja, otthoni telefonja. Ezeket az almezőket a finger használja fel. A program minden egyes mezőhöz az eredeti tartalmat ajánlja fel. Ezt CR leütésével lehet elfogadni, vagy új információ megadásával módosítani, vagy a "none" szóval törölni.

A jelszót illetve a jelszó információkat csak a tulajdonos vagy a super-user változtathatja meg.

Állományok

/etc/passwd	password állomány
/etc/shells	lehetséges shell-ek felsorolása

Lásd még

login(1), finger(1), passwd(5)

M.30 pr - állományok nyomtatáshoz való előkészítése

BSD szintaxis

pr [+n] [-n] [-fmt] [-h fejléc] [-sc] [-wn] [állomány] ...

System V szintaxis

pr [+n] [-n] [-adfmprt] [-eck] [-h fejléc] [-ick]
[-ln] [-nck] [-on] [-sc] [-wn] [állomány] ...

Leírás

A pr egy vagy több állomány tartalmát írja ki a standard kimenetre. Alapértelmezésben a kimenetet lapokra tördeli. Minden lap egy fejléccel kezdődik, amely tartalmazza a dátumot, az állomány nevet (vagy ez utóbbi helyett a megadott fejlécet), valamint az oldalszámot. Ha nem adunk meg állomány argumentumot, akkor a standard bemenetről olvas, így szűrőként használható. A bemeneten érkező FORMFEED karakter a kimeneten változatlanul megjelenik, így a tördelés szükség esetén befolyásolható.

Alapértelmezésben az oszlopok egyenlő szélességűek, és legalább egy szóköz van köztük. Azok a sorok, amelyek nem férnek el, csonkítva lesznek. Ha a -s opciót használjuk, akkor sorok nem csonkítódnak, és az oszlopokat elválasztó karakter a megadott karakter lesz.

Ha a terminál a pr program output-ja, akkor a program futási idejére a write(1) programmal küldött üzenetek megtiltódnak.

A program argumentumait balról jobbra értelmezi. A kapcsolók hatása az argumentumlista végéig, illetve újradefiniálásukig tart.

BSD opciók

-f FORMFEED karaktereket használ a NEWLINE karakterek helyett a lapokra tördeléshez. A program feltételezi, hogy a FORMFEED karakter a lap tetején két üres sort eredményez, így a tényleges papírhossz nem fog változni.

-m Az egyes állományokat egyszerre, egymás melletti oszlopokba írja ki. Pl:

első	második	harmadik
állomány	állomány	állomány
sorai	sorai	sorai

-t Nem írja ki a lap tetejére az 5 soros fejlécet. Ilyenkor még akkor sem fogja a lapokat semmi elválasztani egymástól, ha a -f kapcsolót használjuk.

-h A fejléc argumentumot szövegként értelmezi, és a lapok fejlécében ezen szöveget fogja az állománynév helyett használni.

-ln A lapok hosszát az alapértelmezés szerinti 66 helyett n sorra állítja be.

-sc Az egyes oszlopokat a megfelelő mennyiségű szóköz helyett egyetlen egy "c" karakterrel választja el egymástól. Ha nem adunk meg "c" karaktert, akkor tabulátort helyez el az oszlopok között.

-wn A több oszlopos kimenet támogatása érdekében létezik ezen kapcsoló. Segítségével az oldalak szélessége az alapértelmezés szerinti 72 helyett n karakterre állítható.

-n n oszlopos kimenetet készít. Az oszlopokat nem egyenlíti ki, azaz, ha egy állományban pontosan annyi sor van, amennyi egy oldalra fér, akkor csak egy oszlopot készít. Pl:

Egy	sorait	írja
megadott	három	ki.
állomány	oszlopba	

+n A kimenet kiírását az n. lappal kezdi. Az azt megelőző lapokat elhagyja.

System V opciók

A legtöbb opció hatása ugyanaz, így csak az éltérések soroljuk fel:

-n A többoszlopos kimenet kiegyenlített.

-t A legutolsó lapot nem tölti fel üres sorokkal.

-f Nem tételezi fel, hogy a FORMFEED karakter két üres sort eredményez. Az üres sorokat kiírja, ha kell. Ha a program kimenete a terminál, akkor a -f hatására egy RETURN karakterre vár az első lap kiírása előtt.

-a A -n kapcsolóval együtt használva olyan többoszlopos eredményt kapunk, ahol az állomány egymást követő sorai az egymás mellet levő oszlopokban vannak. Pl:

Egy	megadott	állomány
sorait	három	oszlopba
írja	ki.	

-d Dupla soremelést eredményez.

-p Ha a program kimenete a terminál, akkor minden lap kiírása előtt vár egy RETURN karakterre.

-r Nem ad üzenetet, ha egy állomány üres vagy nem nyitható meg.

-eck A bemeneten minden TAB karaktert a megfelelő számú karakterrel helyettesíti úgy, hogy a tabulátorpozíciók a 2+1, 2*k+1, 3*k+1, stb. pozíciókon legyenek. Ha k 0 vagy nincs megadva, akkor k=8 értéket tételez fel. Ha a c karaktert megadjuk, ami bármilyen nem számjegykarakter lehet, akkor a bemeneten a TAB karakter helyett a c karaktert értelmezi tabulátor karakterként.

-ick A kimeneten szóköz karakterek helyett TAB karaktert használ, ha ez lehetséges. A tabulátor pozíciókat a 2+1, 2*k+1, 3*k+1, stb. pozíciókon tételezi fel. Ha k 0 vagy nincs megadva akkor k=8 értéket használ. Ha a c karaktert megadjuk, ami bármilyen nem számjegykarakter lehet, akkor a kimeneten a TAB helyett a c karaktert használja.

-nck A kimenten minden sort k számjegyből álló sorszámmal lát el. Alapértelmezés szerint k=5. A sorszámok minden oszlop első k+1 pozícióját foglalják el, kivéve a -m kapcsolónál, ahol csak az első oszlopban van sorszám. Ha a c karaktert megadjuk, ami bármilyen nem számjegykarakter lehet, akkor az a sorszám után íródik, mint elválasztó karakter. (Alapértelmezés szerint ez a TAB karakter.)

-ok Offset a sorok kiírásához. Minden kiírt sort k karakterrel eltol jobbra.

Lásd még

cat(1), lpr(1), write(1)

M.31 ps - folyamatok állapotának lekérdezése

Szintaxis

ps [[-]acegjklnrSuUvwx] [-tx]|[num] [kernel_név] [dumpfile] [swapfile]

Leírás

A ps a rendszer folyamatairól ad információkat. Normál esetben csak azok a folyamatok jelennek meg, amelyek effektív felhasználói azonosítója megegyezik a ps programot futtató felhasználó azonosítójával, és az aktuális terminálhoz tartoznak. Ettől eltérő folyamatok is megjeleníthetők a különböző opciók használatával. Az egyes opciók más-más információkat szolgáltatnak. A legfontosabb információk:

- A folyamatot indító terminál neve.
- A folyamat állapota.
- A folyamat tulajdonosának a felhasználó-azonosítója (UID).
- A folyamat tulajdonosának a csoport-azonosítója (GID).
- A folyamat process-ID-je; ezen számmal hivatkozhatunk rá, ha a kill(1) parancsot adjuk ki.
- A folyamat prioritása. (A nagyobb szám jelenti a kisebb prioritást.)
- A folyamat memóriaigénye.
- Azon esemény memóriaképének a kezdőcíme, ami miatt a folyamat vár, vagy "alszik" (sleep);
- A parancs és argumentumai.

Opciók

-a A kapcsoló hatására az összes olyan folyamatról is ad információt, amelyek nem az aktuális felhasználóhoz tartoznak.

-c A parancs nevét a rendszer belső adminisztrációs területéről veszi, és nem a folyamat argumentumlistájáról. Ez utóbbit a folyamat átírhatja.

-e A parancs környezeti változóit (environment) és argumentumait is kiírja.

-g Az összes folyamatról ad információt. Ezen opció nélkül csak az ún. "érdekes" folyamatokról kapunk információt. Nem "érdekes" folyamatok pl. azok, amelyek a user loginra várnak, vagy a login shell.

-j Kiírja az ún. job control információkat is.
-l Hosszú listaformátumot készít.
-r Csak a futó folyamatokat írja ki.
-u Felhasználó orientált eredményt ad.
-U A program privát adatbázisát frissít fel. (Csak a rendszer indításakor van erre szükség.)
-x Azokat a folyamatokat is kiírja, amelyek nem tartoznak terminálhoz.

Lásd még
kill(1), w(1)

Hibák
A folyamatok állapota megváltozhat, amíg a ps fut, ezért a ps csak egy jó közelítését adja a valóságnak.

M.32 pwd - munkakatalógus nevének kiírása

Szintaxis
pwd

Leírás
A pwd kinyomtatja az aktuális munkakatalógus teljes nevét .

Lásd még
cd(1), csh(1), getwd(3)

Hibák
A C shell-ben ez beépített parancs, amitől gyorsabb, de eltér az eredményformátum.

M.33 rm, rmdir - állományok és katalógusok törlése

Szintaxis
rm [-] [-fir] állomány...
rmdir katalógus

Leírás
Az rm törli a megnevezett állományokat mint bejegyzéseket a katalógusból. Ha egy bejegyzés már az utolsó hivatkozás volt az állományra, akkor az állomány tartalma **visszaállíthatatlanul** elveszik. Egy állomány letörléséhez az adott katalógusban írási engedély szükséges, de magához az állományhoz nem kell sem írási, - sem olvasási engedély. Ha a törölni kívánt állományhoz nem rendelkezünk írási joggal, akkor az rm kiírja az állománynevet és védelmi módját, majd beolvas egy - sort a a standard bemenetről. Ha ezen sor "y" betűvel kezdődik, akkor az állományt letörli, egyébként nem. Ezen kérdést nem teszi fel, ha a "-f" kapcsolót megadtuk, vagy ha a standard input nem a terminál.

Az rmdir minden paraméterként megadott katalógust letöröl. Az rmdir csak üres katalógusokat (csak a "."és a ".." van benne) töröl.

Opciók

- Minden további paramétert állománynévnek értelmez. Így lehetséges "-" jellel kezdődő nevet megadni első paraméterként.
-f Üzenet kiírása nélkül letörli az állományt, azaz nem vár megerősítő választ, ha az állomány nem írható.
-i Interaktív mód. Az rm kiírja az összes törölni kívánt állomány nevét és egy kérdőjelet. Ezután egy sort olvas a standard bemenetről. Ha ez a sor y karakterrel kezdődik, akkor letörli az állományt.
-r Rekuzív törlés. Az rm meghívja önmagát minden egyes alkatalógusra és magára a katalógusra is. Így fát is törölhetünk.

Lásd még

ln(1), unlink(2)

M.34 sed - stream szerkesztő

Szintaxis

sed [-n] [-e script] [-f sfile] [állomány]...

Leírás

A sed program a bemenetét astandard outputra másolja, miközben a script paraméternek megfelelő parancsokat végrehajtja minden input sorra. A program működésének részletes leírását az editorokkal foglalkozó E függelék tartalmazza.

Opciók

-n Nem ír ki minden bemenetet a kimenetre automatikusan.
-e script
A script egy parancssorozat, amit a sed végrehajt. Ha csak egy -e opció van, és nincs -f opció, akkor a -e kapcsoló elhagyható.
-f sfile
A parancsokat az sfile nevű állományból veszi.

Lásd még

awk(1), sed(1), grep(1), lex(1), perl(1)

M.35 sort - állományok sorainak rendezése és egybeolvasztása

Szintaxis

sort [-bdfiMnr] [-tc] [sort-mező ...] [-cmu] [-o[ ]eredmény] [-T katalógus]
[-y kmem] [-z recsz] [állomány]...

Leírás

A sort az összes megnevezett állomány sorait együtt sorba rendezi, és a rendezés eredményét kiírja a standard kimenetre. Ha nem adtunk meg bemeneti állományt, vagy ez a "-" karakter, akkor a standard bemenetet használja a program. Így a sort szűrőként is alkalmazható.

Alapértelmezés szerint az összehasonlítandó jelsorozatok teljes sorok, de lehetőség van egy vagy több kulcsmező kijelölésére. Az egyes mezőket a parancs argumentumainak sorrendjében hasonlítja össze. A második, harmadik, stb. mezőt

csak akkor hasonlítja, ha az összes megelőző mező alapján azonosságot talált. Azon sorokat, amelyeket egyformának talált, úgy rendezi sorba, hogy minden byte-ot szignifikánsnak tekint. Ugyancsak alapértelmezés az ASCII kódtábla szerinti lexikális rendezés.

Opciók

A hasonlítást befolyásoló opciók:

-b A kezdő szóközöket és tabulátorokat figyelmen kívül hagyja a mezők kezdetén és végének meghatározásánál.

-d Ún. szótárrendezés. Csak a betűk, számjegyek és szóközök, tabulátorok vesznek részt az összehasonlításban.

-f A kis- és nagybetűket nem különbözteti meg.

-i Figyelmen kívül hagy minden nem ASCII karaktert.

-M Hónapsorrend. A mező első három nem szóköz karaktere nagybetűvé konvertálódik, és így hasonlítja a hónapok angol hárombetűs rövidítéséhez (JAN FEB ... DEC) a mezőt. Az intervallumból kieső mezők a JAN elé kerülnek. Az opció bekapcsolja a -b opciót.

-n Numerikus sorrend. A mező elején álló számsorozatot, ami mínusz jelet és tizedespontot is tartalmazhat, számnak értelmezi, és az aritmetikai értéke szerint rendezi. Az opció bekapcsolja a -b opciót, ha legalább egy mezőt adtak meg a parancssorban.

r Megfordítja az aktuális rendezési szempontot.

Mezőket meghatározó opciók:

tc A mezők közti tabulátor karakter a c karakter lesz.

sort-mező

Ezzel a sor egy mezőjét jelölhetjük ki, mint rendezési kulcsot. A sort_mező a következő két megadási mód egyikével adható meg:

+sw[cf]

+sw -ew[cf]

Ahol az sw a sor azon szava, amelyet először tartalmaz az adott mező, ew pedig az, amely már nem tartozik a mezőhöz (a szavak számozása 0-tól kezdődik). Minden mezőhöz megadható az értelmezése ill. rendezési szempontja a cf karaktersorozat segítségével, ami a fenti kapcsolók kombinációja lehet ("-" jel nélkül). Ez természetesen csak az adott mezőre érvényes, és felüldefiniálja a parancs argumentumában megadott, egyébként az egész sorra érvényes kapcsolókat. Ha a -ew kapcsoló nincs megadva a meződefinícióban, akkor a mező a sor végéig tart.

Az sw ill. ew pozíciókhoz karakter offset is megadható, ami azt jelenti, hogy az adott mező a szó megfelelő karakterénél kezdődik ill. végződik. Ennek formája a w.c forma, ahol c jelzi a pozíciót a szón belül (a szó első karakterére c=0). Ha ab -b kapcsoló be van kapcsolva, akkor c az első nem space vagy tabulátor karaktertől számítódik, egyébként a határoló karakter is beszámít c-be.

Egyéb opciók:

-m Ellenőrzi, hogy az input megfelelően rendezett-e. Nem ad eredményt, kivéve ha a rendezés nem felelt meg a magadott szabályoknak.

-m Csak összefésüli a különböző inputokat, ezek mindegyikét már sorbarendezettnek tekinti.

-u A rendezés után ismétlődő sorokból csak egyet-egyet ír ki az eredménybe.

-oeredmény
-o eredmény
A következő argumentumot a kimeneti állomány nevének tekinti, amelyet a standard output helyett fog használni. Ez az állomány a bemeneti állományok bármelyike is lehet, kivéve, ha a -m (összefésülő) opciót kívánjuk használni.
-y kmem
A program által kezdetben lefoglalt memória méretét adhatjuk meg kilobyte-okban. A -y0 kezdetben a legkisebb memóriát foglalja, majd úgy növeli, ahogyan szükséges. A kapcsoló nélkül a program a legkisebb állományhoz is lefoglalja a rendszerben használt alapértelmezés szerinti méretű memóriát.
-z recsz
A leghosszabb input sor méretet lehet megadni.
-T katalógus
A megadott katalógust használja az ideiglenes állományok létrehozásához.

Példák

A következő parancs a file tartalmát rendezi a második szó szerint:

sort +1 -2 file

A következő parancs a file1 és a file 2 tartalmát rendezi visszafelé a második szó első nem space ill. nem tab. karaktere szerint:

sort -r +1.0b -1.1b file1 file2

A következő parancs a /etc/passwd állományt rendezi a numerikus felhasználói azonosító alapján:

sort -t: +2 -3n /etc/passwd

A következő parancs a /etc/passwd állományt rendezi a numerikus csoportazonosító, azon belül a numerikus felhasználói azonosító alapján.

sort -t: +3 -4n +2 -3n /etc/passwd

Lásd még

comm(1), join(1), rev(1), uniq(1)

M.36 split - állomány darabokra tördelése

Szintaxis

split -n [állomány [név]]

Leírás

A split beolvassa a paraméterként kapott állományt, és kiírja "n" soros darabokban (alapértelmezés szerint n = 1000) különböző output állományokba (pontosan annyiba, ahányra szükség van). Az első eredményállomány neve az "aa" toldalékkal ellátott név lesz, majd így tovább, abc sorrend szerint. (Pl: névaa, névab, névac stb.) Ha név paramétert nem adunk meg, akkor "x"-et használ helyette. (xaa, xab, xac stb.)

Ha bemenő állományt nem adunk meg, vagy a "-" argumentum szerepel a helyén, akkor a standard bemenetről olvas.

Opciók

-n Az egyes darabokban levő sorok száma.

M.37 strings - látható karakterekből álló stringek keresése

Szintaxis

strings [-] [-o] [-n] [állomány] ...

Leírás

A program a megadott állomány olyan 4 vagy annál hosszabb, látható karakterekből álló karaktersorozatait írja ki a standard kimenetre, amelyek NEWLINE vagy NUL karakterre végződnek. (Minden karakter látható, aminek ASCII kódja a 040-177 tartományban van.)

Opciók

- `-` Mindenütt keres. Ha ez az opció nincs megadva, akkor a strings csak a kezdőértékezett adatterületén keres object és bináris állományokban.
- `-o` Kiíráskor kiírja a megtalált karaktersorozat címét is.
- `-n` A kapcsolóval a kiírandó karaktersorozat minimális hosszát lehet megváltoztatni 4-ről az n helyére írt hosszra.

M.38 tail - utolsó néhány sor kiíratása

Szintaxis

tail [+|-szám[lbc]][rf] [állomány]

Leírás

A tail program a megadott állomány, vagy a standard bemenet utolsó néhány sorát írja ki a standard kimenetre.

Ha +szám argumentum van megadva, akkor a kiírás a megadott állomány elejéhez képest a megadott sorban kezdődik, és az állomány végéig tart.
Ha -szám argumentum van megadva, akkor a kiírás a megadott állomány végéhez képest a megadott sorban kezdődik, és az állomány végéig tart.
A szám értelmezése az l, b, c módosítótól függően sor, blokk vagy karakter mértékegységben történik. Ha nincs megadva, akkor sor az alalpértelmezés.

Az r módosító hatására a tail fordított sorrendben írja ki a sorokat, az f pedig azt eredményezi, hogy a program nem áll le az állomány végénél, hanem végtelen ideig próbálkozik tovább olvasni. Ha ilyenkor az állomány növekszik, akkor ettől kezdve minden sort kiír.

Lásd még

dd(1), head(1)

M.39 tar - archívum kezelő segédprogram

Szintaxis

tar [-][0-9][cxtpvbf] [blokkolási_faktor] [archívum] állomány ...

Leírás

A tar program segítségével lehetséges az ún. tar formátumú állományok kezelése. Eredetileg szalagos archívum kezelésre készült a program, de mivel a UNIX-ban a perifériák ugyanúgy kezelhetők, mint a normál állományok, ezért más eszközök és normál állományok is kezelhetők a programmal. Ha a program `állomány` paramétereként katalógust adunk meg, úgy az adott katalógusból kiinduló teljes katalógusstruktúra (részfa) archiválható. Fontos azonban tudni, hogy ha az archívum létrehozásakor az archiválandó állományokat teljes útnévvel adjuk meg, akkor azok csak az útnévvel adott eredeti helyükre tölthetők vissza.

Sajnos a programnak meglehetősen sok változata létezik, melyek hasznos kiegészítésekkel rendelkeznek. Ilyen pl. az include, exlude lista megadásának lehetősége, melyeket nem minden változat támogat.

Opciók

c Archívum létrehozása a megadott állományokból.

x Archívum kibontása, ill. a megadott állományok visszatöltése.

t Archívum tartalmának listázása.

p Visszatöltéskor az archívumban levő védelmi biteket veszi figyelembe, és figyelmen kívül hagyja az aktuális umask értéket. Alapértelmezés szerint a tar programot futtató felhasználó tulajdonába kerülnek a visszatöltött állományok, kivéve ha a felhasználó a super-user, ekkor a p kapcsoló alapértelmezés szerint be van kapcsolva, így a setuid és setgid bitek, valamint a tulajdonos is az archívumban levő érték lesz.

v Ún. fecsegő opció. részletesebb listát ad, valamint kiírja annak az állománynak a nevét amelyikkel éppen foglalkozik.

b A következő argumentum egy ún. blokkolási faktor, amely megadja, hogy egy mágnesszalag blokk hányszor 512 byte-os legyen. Alapértelmezés szerint 20 a blokkolási faktor.

f A következő argumentum egy állomány neve, amely megadja az archívum állomány nevét. Ellenkező esetben a rendszer alapértelmezés szerinti mágnesszalag egységet tekinti archívumnak. Az állomány neve helyett, sok más UNIX segédprogramhoz hasonlóan '-' jel is állhat, ami értelemszerűen a standard input-ot, vagy a standard output-ot jelenti.

A program **cxt** kapcsolói közül értelemszerűen egyszerre csak az egyik szerepelhet.

A következő kapcsolókat nem minden tar változat értelmezi:

h Szimbolikus link követése. Alapértelmezés szerint a tar nem írja fel a szimbolikus link által hivatkozott állományok tartalmát, csupán a hivatkozás tényét.

i Visszatöltéskor figyelmen kívül hagyja a kontrolszumma hibákat.

I A következő argumentum egy ún. include állomány neve, mely tartalmazza az archívumba felírandó állományok nevét.

X A következő argumentum egy ún. exclude állomány neve, mely azokat az állományneveket tartalmazza amelyeket nem kell az archívumba felírni.

Példák

A következő parancs a MUNKA katalógus tartalmát, és annak alkatalógusait menti a munka.tar állományba:

```
tar -cvf munka.tar MUNKA
```

Érdekes és egyben fontos lehetőséget ad a tar program egy részfa lemásolására. Ha ugyanis a `cp -R /src /dst` parancsot alkalmazzuk, akkor az esetleges linkeket elveszítjük, nem beszélve az állományok módosítási idejéről. A fenti feladatot a tar segítségével a következőképpen oldhatjuk meg.

```
cd /src
tar -cf - . | ( cd /dst; tar -xf - )
```

vagy ugyanez biztonságosabban:

```
cd /src
tar -cf - . | ( cd /dst && tar -xf - )
```

Ugyanis, ha a cd valamiért nem sikerülne, úgy az első esetben a tar tönkre tenné a forrásállományokat.

Még izgalmasabb feladat lehet egy részfának egy távoli gépről a helyi gépre történő másolása tar segítségével:

```
rsh távoli_gép 'tar -cf - MUNKA' | tar -xvf -
```

Lásd még

ar(1), cpio(1), mt(1), pax(1)

M.40 tee - standard kimenet másolása

Szintaxis

tee [-ai] [állomány] ...

Leírás

A tee lemásolja a standard bemenetet a standard kimentre, és a megnevezett állomány(ok)ban is készít egy másolatot a standard bemenetéről.

Opciók

-a A kimenetet a meglevő állomány(ok)hoz másolja. Egyébként törli a meglevő állományok korábbi tartalmát.

-i Figyelmen kívül hagyja a megszakításokat.

M.41 telnet - felhasználói felület a telnet protokollhoz

Szintaxis

telnet [host [port]]

Leírás

A telnet program segítségével lehet kommunikálni másik hosttal a TELNET protokollon keresztül. Ha a telnetet paraméterek nélkül hívjuk, akkor parancs módba kerül, amit a "telnet>" prompt jelez. Ebben a módban az alább felsorolt parancsokat értelmezi és hajtja végre. Ha argumentumokkal hívjuk, akkor rögtön egy open parancsot (lásd később) hajt végre a megadott argumentumokkal.

Ha egy kapcsolat felépült, a telnet input módba kerül. Ebben a módban a begépelt szöveget elküldi a másik hostnak. Az input mód lehet ún. raw mód vagy nem raw mód, attól függően, hogy a távoli rendszer melyiket támogatja.

Raw módban a begépelt karakterek azonnal továbbítódnak feldolgozásra a hosthoz.

Nem raw módban a gépelt szöveget lokálisan megjeleníti (echo), és csak a már kész szövegsorokat küldi el a hostnak. A "local echo character" (kezdetben a "^E") használható arra, hogy ki illetve bekapcsoljuk a lokális megjelenítést (erre rendszerint a jelszavak láthatatlan begépelésénél lehet szükség).

Mindkét módban, ha a localchars kapcsoló IGAZ, a felhasználó quit, intr, és flush karaktereit a telnet lokálisan észreveszi, és a TELNET protokollon keresztül továbbítja a távoli hostnak.

Telnet kapcsolat közben a telnet program parancs módba kapcsolható a telnet "escape character" (^]) begépelésével. Parancs módban a hagyományos sorszerkesztő funkciók használhatók.

Használat

Telnet Parancsok

A következő parancsok használhatók. (Elegendő mindig a parancsok és argumentumaik annyi karakterét begépelni, amely az egyértelmű azonosításhoz szükséges.)

open host [port]
: Felépíti a kapcsolatot a megnevezett hosttal. Ha nem adunk meg port számot, a telnet az alapértelmezés szerinti porton keresztül kísérli meg felvenni a kapcsolatot a távoli TELNET szerverrel. A host megadása történhet névvel és számmal leírt Internet címmel is.

close
: Lebontja a telnet kapcsolatot a hosttal és parancsmódba lép.

quit
: Lebontja a telnet kapcsolatot és befejezi a működést. Parancsmódban begépelt EOF hatása ugyanez.

z
: Felfüggeszti a működést. Ez a parancs csak akkor működik, ha a használt shell támogatja a job control funkciót.

mode type
: A type lehet "line" raw módhoz, vagy "character" nem raw módhoz. A telnet csak a távoli host beleegyezése után lép az új módba.

status
: A telnet aktuális státuszát mutatja. (kiépített kapcsolat, mód)

display [argument ...]
: Megjeleníti az összes vagy a megadott kapcsolók és beállítások értékét. (lásd később)

? [command]
: Segítség (help) kérés. Argumentumok nélkül a telnet egy összefoglalót ad. Ha megadunk egy parancsot, akkor a telnet az arra a parancsra vonatkozó információkat írja ki.

send arguments
: Elküld egy vagy több speciális karaktersorozatot a távoli hostnak. A következő argumentumokat lehet megadni (egyszerre akár többet is):

escape
Elküldi az aktuális telnet "escape charactert" (kezdetben ez a `^]').
synch
Elküldi a TELNET SYNCH sorozatot. Ennek hatására a távoli rendszer eldobja az összes korábban begépelt (de még fel nem dolgozott) inputot.
brk Elküldi a TELNET BRK (Break) sorozatot.
ip Elküldi a TELNET IP (Interrupt Process) sorozatot, melynek hatására a távoli host abortálja az éppen futó folyamatot.
? Kiírja a send parancsra vonatkozó összes információt.

set argument value
Beállítja az adott telnet változót az adott értékre. Ha a value "off", akkor kikapcsolja a változóval kapcsolatos funkciót. A következő változókat lehet megadni:
echo
Ez az a karakter, mellyel nem raw módban a lokális kijelzést (echo) be ill. kikapcsolhatjuk.
escape
A telnet "escape character", melynek begépelésével parancs módba kapcsolhatunk.
interrupt
Ha a telnet localchars módban van (lásd alább) akkor az interrupt karakter begépelésével egy TELNET IP sorozatot küldhetünk a távoli hostnak. Az interrupt karakter kezdeti értéke a terminál intr karakterevel egyezik.
quit
Ha a telnet localchars módban van (lásd alább) akkor a quit karakter begépelésével egy TELNET BRK sorozatot küldhetünk a távoli hostnak. A quit karakter kezdeti értéke a terminál quit karakterével egyezik még.

toggle arguments
Különböző, a telnet működését szabályozó flageket lehet ki/bekapcsolni. A következő flagek lehetségesek: (egyszerre többet is megadhatunk.)
localchars
Ha ennek állapota IGAZ, akkor a flush, interrupt, quit, stb. karaktereket a telnet lokálisan felismeri, a megfelelő TELNET sorozattá konvertálja és elküldi a távoli hostnak. A kezdeti értéke raw módban HAMIS, nem raw módban IGAZ.

crmod
A RETURN módot kapcsolja. Ha ez a mód engedélyezve van, akkor a távoli hosttól jövő RETURN karaktereket egy RETURN, LINEFEED karakterpárrá alakítja. Nincs hatással a felhasználó által begépelt karakterekre. Kezdeti értéke HAMIS.
debug
Socket szintű debug opció. Csak a rendszerprogramozók, vagy a superuser számára hasznos. Kezdeti értéke HAMIS.
netdata

A hálózaton érkező adatokat hexadecimális formában írja ki. Kezdeti értéke HAMIS.

? Kiírja az összes használható flaget.

Lásd még

csh(1), rlogin(1), stty(1), inet(3), hosts(5)

Hibák

A számos beállítható paramétert jó lenne egy .telnetrc file-ban tárolni.

M.42 test - feltételes kifejezés kiértékkelő parancs

Szintaxis

test kifejezés

Leírás

A megadott kifejezést kiértékeli és ha az igaz, akkor nulla visszatérési értékkel tér vissza, egyébként nem nullával. Ha nincs paramétere, akkor nem nulla értékkel tér vissza.

Lásd még

sh(1), find(1)

M.43 time - egy parancs végrehajtási ideje

Szintaxis

time [parancs]

Leírás

A megadott parancsot végrehajtja, majd megadja a parancs elindításától a megállásáig eltelt időt (real), a rendszerben eltöltött időt (system), és a parancs tényleges végrehajtásával eltöltött időt (user).

C shell esetén a time beépített parancs. Ekkor a time parancs paraméter nélkül elindítva megadja az aktuális shell, és az összes gyereke által felhasznált időt, azaz az összes elindított parancs által felhasznált időt.

Lásd még

csh(1)

Hibák

Az összes eltelt időt másodpercben méri, de a CPU idő 1/50-ed másodpercben van mérve. Így CPU idő majdnem egy másodperccel is nagyobb lehet, mint az eltelt idő.

A /bin/time program és a C shell-be beépített time parancs eredményformátuma nem azonos. A System V rendszerben használt /bin/time program egy harmadik eredményformátumot használ.

M.44 tr - karakter átalakítás

BSD szintaxis

tr [-cds] [string1 [string2]]

BSD leírás

A tr a standard bemenetet lemásolja a standard kimenetre, miközben a kiválasztott karaktereket letörli vagy kicseréli. A string1 karaktersorban talált karaktereket átalakítja a string2 karaktersor megfelelő pozícióján talált karakterre. Ha a string2 rövidebb, akkor az kiegészítődik a string1 hosszára, mégpedig úgy, hogy a string2 utolsó karakterét ismétli a megfelelő számszor.

Mindkét karaktersorozatban lehetőség van intervallum kijelölésére. Pl. az A-G az összes karaktert jelenti A és G között növekvő sorrendben. Nem látható karakter is megadható a "\" után 1, 2 vagy 3 oktális számjeggyel.

System V leírás

Ha a string2 rövidebb, akkor az nem egészítődik ki, hanem srting1 rövidül a string2 hosszára. Mindkét karaktersorozatban a következő konvenciók alkalmazhatók karakterosztályok vagy ismétlődő karakterek megadására:

[a-b] Az összes olyan karaktert jelenti, amelyeknek ASCII kódja az "a" karakter - és a "b" karakter kódja közé esik a határokat is beleértve.

[a*N Az "a" karakter N-szeres ismétlődését jelenti, ahol N egy egész szám, vagy elmarad. N attól függően oktális vagy decimális, hogy első számjegye a "0", vagy nem. Ha N nulla, vagy elmarad, akkor tetszőleges nagy lehet. Ezt a lehetőséget a kitöltő karakterek megadására lehet használni.

Opciók

-c Komplementálja a string1-ben a karaktereket. (A karakterek kódja 01-től 0377-ig tartó intervallumban van.)

-d Minden string1-ben levő karaktert töröl az inputról.

-s Az eredményben ismétlődő minden olyan karaktert, ami a string2-ben előfordul, egyetlen egy karakterre cserél le.

A fenti opciók tetszőleges kombinációja használható.

Példák

A következő példa segítségével a file1-ben található összes szót elhelyezi a file2 soraiba, mindegyiket külön sorba. Szónak minősül az alfabetikus karakterek leghosszabb string-je. A stringeket idézőjelek közé zártuk, mert egyes karaktereit a shell maga is értelmezné; 012 az újsor jel oktális kódja.

```
tr -cs "[a-Z][a-z]" "[\\012*]" <file1 >file2
```

Lásd még

sh(1), ed(1), ascii(7)

Hibák

A program sem string1-ben, sem string2-ben nem képes az ASCII NUL karaktert kezelni. Ezt a karaktert a bemenetről mindig törli.

M.45 uniq - állomány ismétlődő sorainak kezelése

Szintaxis

uniq [-cdu [+n] [-n]] [input [output]]

Leírás

A uniq beolvassa az input állomány (hiányában a standard bemenet) sorait, és összehasonlítja az egymás utáni sorokat. Normál esetben az ismétlődő sorok második és további példányát elhagyja, és így írja ki az output állomány-ba (hiányában a standard kimenetre).

Opciók

-c A kapcsoló felüldefiniálja -u és -d kapcsolókat, és alapértelmezés szerinti kimenetet állít elő, de minden sort megelőz egy szám, amely azt jelzi, hogy hány példányban szerepelt az adott sor.

-d Csak az ismétlődő sorok egy-egy példányát írja ki.

-u Csak azon sorokat írja ki, amelyek nem ismétlődtek meg.

Az alapértelmezés a -u és -d kapcsoló uniója. Az n argumentumok segítségével a sorok elejét lehet átlépni:

-n Az első n mezőt, valamint az összes előttük illetve köztük szereplő szóközt figyelmen kívül hagyja. Mezőnek számít minden olyan karaktersorozat, amelyet szóköz vagy tabulátor határol.

+n Az első n karaktert figyelmen kívül hagyja. A mezőket a karakterek elhagyása előtt lépi át, ha a "-n" kapcsoló is szerepel.

Megjegyzés

Az ismétlődő soroknak egymás után kell állniuk, hogy megtalálja őket, lásd sort(I).

Lásd még

sort(1), comm(1)

M.46 vi - képernyő-orientált editor

Szintaxis

vi [-tag] [-r] [+parancs] [-l] [-wn] [állomány]...

Leírás

A vi program egy képernyő-orientált editor. A program működésének részletesebb leírását az editorokkal foglalkozó E függelék tartalmazza.

Lásd még

ed(1), ex(1)

M.47 wc - kiírja a sorok, szavak és karakterek számát

Szintaxis

wc [-lwc] [állomány] ...

Leírás

A wc megszámolja a megnevezett állományok sorait és szavait. Ha nem adunk meg állománynevet, akkor a standard inputról olvas. A "szó" fogalmán egymástól szóköz, tabulátor vagy újsor karakterekkel elválasztott látható karakterek leghosszabb

sorozatát értjük. Minden más karaktert a program egyszerűen figyelmen kívül -hagy.

Opciók

-l Sorokat számol.
-w Szavakat számol.
-c karaktereket számol.

Alapértelmezés szerint asz összes kapcsoló be van kapcsolva (-lwc).

M.48 who - ki van bejelentkezve a rendszerbe

Szintaxis

who [who_file] [am i]

Leírás

A who program paraméter nélkül indítva kinyomtatja a rendszerbe bejelentkezettek nevét, a terminál számát és a bejelentkezés idejét. Ilyenkor a "/etc/utmp" állományban tárolt információkat olvassa ki. Ha a who_file paramétert megadtuk, akkor /etc/utmp helyett a megadott állományt használja.

A who program két argumentummal indítva ugyanúgy dolgozik, mintha argumentum nélkül hívtuk volna meg, de csak azon sort nyomtatja ki, amely azon terminálra igaz, amelyiken meghívták. Így a "who am i" parancs megmondja, hogy ki jelentkezett be az adott terminálon. (Természetesen ugyanígy a "who x y" is ezt eredményezi.)

Lásd még

login(1), w(1), whoami(1)

M.49 write - üzenet a másik felhasználónak

Szintaxis

write user [ttyname]

Leírás

A write a standard inputról sorokat másol a megadott felhasználó terminájára. Amikor a programot elindítjuk, akkor a partner terminálján megjelenik a következő üzenet:

Message from login_név@gép on tty at time ...

Ahol a login_név, gép, és a tty az üzenetet kezdeményező felhasználó adatai. Erre a partner célszerűen szintén egy write paranccsal válaszol. A write állomány végéig olvas a bemenetről. Ha véget ért a bemenet (terminálról ez CTRL-D), akkor a partner képernyőjén kiírja az EOT üzenetet, és a program megáll.

Ha olyan felhasználónak akarunk üzenni, aki egyszerre több terminálon is be van jelentkezve, akkor a ttyname paraméterrel választhatunk a terminálok közül.

A wirte-tal történő üzenetküldés megtiltható a mesg(1) paranccsal.

Ha az input sor elején ! áll, akkor write meghívja a sor maradék részével a shell-t.

Lásd még

mail(1), mesg(1), talk(1), who(1)

Tárgymutató

1. Az M függelékben szereplő parancsok betűrendes mutatója