

Információs rendszerek üzemeltetése – Linux admin –

BME TMIT

2017

| | Oldalszám |
|---|-----------|
| 0. Bevezetés | 2 |
| 0.1. Fontos tudnivalók | 2 |
| 0.2. A rendszer elindítása | 4 |
| 1. A rendszer indítása után... | 5 |
| 1.1. Feladatok | 5 |
| 2. Az Apache 2 webkiszolgáló | 7 |
| 2.1. Az Apache biztonsági beállításai | 10 |
| 2.2. Váltás Apache 2.4-re | 11 |
| 2.3. A .htaccess fájl | 12 |
| 2.4. Feladatok | 13 |
| 3. Linux héjprogramozás | 15 |
| 3.1. A <i>cut</i> | 17 |
| 3.2. A <i>date</i> parancs | 17 |
| 3.3. A <i>bash</i> | 17 |
| 3.4. Karakterfolyamok kezelése | 25 |
| 3.5. A <i>patch</i> | 32 |
| 3.6. Feladatok | 34 |
| 4. A Monit | 38 |
| 4.1. A monit indítása | 38 |
| 4.2. Az első lépések | 39 |
| 4.3. Konfigurációs mintapéldák | 40 |
| 4.4. Feladatok | 41 |
| Hivatkozások | 43 |
| 5. Appendix | 44 |

0 — Bevezetés

Ez a labormérés négy témakörön és rengeteg feladaton keresztül próbál képet adni a Linux adminisztrátorok világáról. De nem csak képet szeretne adni, hanem lehetőséget adni picit elmerülni ebben a furcsa világban. Éppen ezért ez a labormérés jelentős interaktivitást kíván meg: tipikusan minden feladatnál a Linux-os man oldalakat, Internetes keresőt kell használni. A feladatok után megadott támogató megjegyzések nem a megoldások, csak iránymutató segítségek – nem elég ezeket begépelni.

Fontos tudnivalók

A segédletben szereplő összes feladat tökéletes megoldása nem kötelező, de minden témakörből legalább 40%-ot el kell érni a labor sikeres teljesítéséhez. A labor sikeres elvégzésébe csak a ♠ szimbólummal jelölt feladatok számítódnak bele, a többi feladat amolyan ráhangolódás, illetve hasznos segítséget jelent a jelölt feladatok megoldásához. Így, természetesen, a nem jelölt feladatok megoldása nem kötelező, de hasznos lehet. Az opcionálisként jelölt feladatok megoldása sem kötelező, de ezen feladatok beleszámítanak a végső értékelésbe, mégpedig oly módon, hogy pontszámuk hozzáadódik azon témakör pontszámához, amelyből nem sikerült 40%-ot elérni.

Amennyiben egy feladat „ábécézve” van akkor az *A*, *B*, *C*, *D* részfeladatok közül nem kell mindet megoldani, csak azt, amelynek a kódja az Ön NEPTUN kódjából származtatható. Ehhez csak a NEPTUN kód karaktereinek a *ASCII* kódját (lásd 1. ábra) kell összegezni, majd venni a négyvel vett maradékát. Amennyiben a maradék *i*) nulla, akkor Önnek az *A* jelű feladatot, amennyiben *ii*) egy, akkor a *B* jelű feladatot, amennyiben *iii*) kettő, akkor a *C* jelű feladatot, és végezetül amennyiben *iv*) három, akkor a *D* jelű feladatot kell megoldani.

A feladatok sikeres megoldását egy (bash) héjprogram a labor végén automatikusan ellenőrzi, ennek elindításához mérésvezetői segédlet szükséges. A program nem csak ellenőrzi és értékeli a feladatokat, hanem magát a labor elvégzését bizonyító ellenőrző kóddal ellátott jegyzőkönyvet is legenerálja (és majd ezt a fájlt kell elküldeni az iru.bme@gmail.com címre). Sajnos egy „buta” program az emberi szemhez képest kevésbé elnéző, ezért a következő szabályok betartása gyakorlatilag elengedhetetlen a labor sikeres elvégzéséhez:

- Mindig használják a feladatokban megadott felhasználói neveket és jelszavakat! A jelszó az esetek döntő többségében az Önök NEPTUN kódja lesz, ezt írják mindig ugyanabban a formátumban (azaz ha a NEPTUN kód „al-mafa”, akkor az használható „AlMafA”-ként is, de akkor végig csak ebben a formában).

| | | | | | | | | | | | | | | | |
|----|-----|----|-----|----|-----|----|---|----|---|----|---|-----|---|-----|-----|
| 0 | NUL | 16 | DLE | 32 | SPC | 48 | 0 | 64 | @ | 80 | P | 96 | ` | 112 | p |
| 1 | SOH | 17 | DC1 | 33 | ! | 49 | 1 | 65 | A | 81 | Q | 97 | a | 113 | q |
| 2 | STX | 18 | DC2 | 34 | " | 50 | 2 | 66 | B | 82 | R | 98 | b | 114 | r |
| 3 | ETX | 19 | DC3 | 35 | # | 51 | 3 | 67 | C | 83 | S | 99 | c | 115 | s |
| 4 | EOT | 20 | DC4 | 36 | \$ | 52 | 4 | 68 | D | 84 | T | 100 | d | 116 | t |
| 5 | ENQ | 21 | NAK | 37 | % | 53 | 5 | 69 | E | 85 | U | 101 | e | 117 | u |
| 6 | ACK | 22 | SYN | 38 | & | 54 | 6 | 70 | F | 86 | V | 102 | f | 118 | v |
| 7 | BEL | 23 | ETB | 39 | ' | 55 | 7 | 71 | G | 87 | W | 103 | g | 119 | w |
| 8 | BS | 24 | CAN | 40 | (| 56 | 8 | 72 | H | 88 | X | 104 | h | 120 | x |
| 9 | HT | 25 | EM | 41 |) | 57 | 9 | 73 | I | 89 | Y | 105 | i | 121 | y |
| 10 | LF | 26 | SUB | 42 | * | 58 | : | 74 | J | 90 | Z | 106 | j | 122 | z |
| 11 | VT | 27 | ESC | 43 | + | 59 | ; | 75 | K | 91 | [| 107 | k | 123 | { |
| 12 | FF | 28 | FS | 44 | , | 60 | < | 76 | L | 92 | \ | 108 | l | 124 | |
| 13 | CR | 29 | GS | 45 | - | 61 | = | 77 | M | 93 |] | 109 | m | 125 | } |
| 14 | SO | 30 | RS | 46 | . | 62 | > | 78 | N | 94 | ^ | 110 | n | 126 | ~ |
| 15 | SI | 31 | US | 47 | / | 63 | ? | 79 | O | 95 | _ | 111 | o | 127 | DEL |

1. ábra. ASCII kódok

- Amennyiben futtatható héjprogramot, szkriptet kell írni, akkor annak nevét származtassa a feladat sorszámából az alábbi módon:

7.4.A feladat → *7-4.sh*,

azaz a feladat sorszámában szereplő pontokat egyszerűen cserélje le alulvonásokra és illessze a végére az *sh* végződést. A programokat másolja minden esetben a *laboruser* felhasználó *bin* könyvtárába (*/home/laboruser/bin*). Amennyiben ez a könyvtár nem létezik, akkor hozza létre. Ne feledjen futtatási jogokat adni a fájlnak!

- Minden esetben pontosan kövesse a megadott specifikációkat!

A segédlet végén igyekeztünk összegyűjteni a labor sikeres elvégzéséhez nélkülözhetetlennek vélt és erősen ajánlott könyvek, jegyzetek listáját. De ha mégis választanunk kellene, akkor mi a következő könyveket ajánlanánk:

1. Tony Bautts, Terry Dawson, Gregor N. Purdy: Linux hálózati adminisztrátorok kézikönyve,
2. Arnold Robbins and Nelson H. F. Beebe: Classic Shell Scripting,
3. Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley: Unix and Linux System Administration Handbook és
4. Lars Wirzenius, Joanna Oja, Stephen Stafford, Alex Weeks: Linux rendszer-adminisztrátorok kézikönyve.

A könyveken és jegyzeteken túl hiperhivatkozásokkal is kibővítettük az ajánlott irodalmak listáját. Ezek mindegyike egy-egy témakör elmélyítését segítheti elő, egy-egy trükköt vagy módszert szemléltetnek, mutatnak be. Ezek közül szeretnénk kiemelni az Ubuntu Linux hivatalos dokumentációját, ahol számos problémára hasznos tanácsot kaphatunk, és amely megoldások könnyen átemelhetők más Debian alapú Linux disztribúciókba is.

Végezetül nincs más hátra, mint hogy eredményes labor és sok sikert kívánjunk Önöknek!

A rendszer elindítása

A VM elindítása

VMWare playerben indítsa el a virtuális gépet! Amennyiben az feldob egy ablakot, melyben a VM eredetére kíváncsi, válassza az „I copied it” opciót! A gazda rendszerbe visszatérni a Ctrl+Alt kombinációval lehet.

Hozzáférési adatok

A virtuális gépre két felhasználó áll rendelkezésre:

felhasználói név: **root**

jelszó: **irulabor**

felhasználói név: **laboruser**

jelszó: **laboruser**

A virtuális gépre *laboruser*ként jelentkezzen be!

1 — A rendszer indítása után...

Feladatok

1.1. feladat: Nézze meg, hogy a rendszeren mely szolgáltatások várnak hálózati kapcsolatokra (figyelnek adott porton), és azok mire valók!

`man netstat, netstat -l`

1.2. feladat: Nézze meg, mi a futtatott Debian rendszer kódneve!

`lsb_release -da`

1.3. feladat: Írassa ki a telepített csomagokat! Telepítve van a Midnight Commander fájlkezelő a rendszerben?

`dpkg -l`

♠ **1.4. feladat:** A céges policy-k szerint a távoli bejelentkezést biztosító szolgáltatásoknak (ssh) csak a lokális hálózatról kell elérhetőnek lenniük. Módosítsa a tűzfalszabályokat ennek megfelelően!

`iptables -L INPUT, iptables -A INPUT ...`

♠ **1.5. feladat:** A céges policy-k szerint a gépnek nem szabad válaszolnia a pingelésre. Módosítsa ennek megfelelően a tűzfal szabályait!

♠ **1.6. feladat:** Adjon hozzá a rendszerhez egy új felhasználót, akinek a neve legyen „mekkelek”, jelszava pedig az Ön NEPTUN kódja! Nézze meg, milyen új bejegyzés született az `/etc/passwd` fájlban!

`adduser`

♠ **1.7. feladat:** Tegye lehetővé az új felhasználó számára, hogy rendszergazda jogokkal futtathasson minden programot!

`visudo`

♠ **1.8. feladat:** A biztonság érdekében tiltsa le a `root` felhasználó SSH-n történő bejelentkezésének jogát!

`mc, /etc/ssh/sshd_config, /etc/init.d/ssh restart`

1.9. feladat: Valósítson meg RSA kulcsokkal működő autentikációt a fizikai és a virtuális gép között a saját, újonnan létrehozott (mekkelek) felhasználó számára! Ehhez az `ssh-keygen` program segítségével hozzon létre egy új publikus-privát kulcspárt. A létrehozott kulcspár privát részét másolja át a gazdagépre, majd a `puttygen` program segítségével konvertálja át a `PuTTY` számára emészthető formátumba. Végezetül a virtuális gépen a

```
1 $ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

parancs kiadásával engedélyezze Mekk Elek RSA bejelentkezését! Tesztelje le az új lehetőséget!

♠ **1.10. feladat:** Telepítse a MySQL-t a guest gépre! Az adminisztrátor jelszava az egyszerűség kedvéért legyen „root”! Természetesen éles környezetben nehezen kitalálható jelszavakat kell használni. Nézze meg, hogy a szerver melyik verzióját sikerült installálnia!

```
apt-get install
```

1.11. feladat: Vizsgálja meg a MySQL szerver beállításait (*/etc/mysql*)! Derítse ki, hogy milyen porton figyel a MySQL szerver, és hol tárolja az adatbázisokat.

1.12. feladat: A *netstat* parancs segítségével állapítsa meg, hogy fogad-e kéréseket a MySQL szerver, és ezt TCP vagy UDP protokollon teszi-e, továbbá hogy elérhető-e a MySQL szolgáltatás a képen kívülről, tehát a helyi hálózatról, vagy bármilyen IP címről!

```
netstat
```

1.13. feladat: Jelentkezzen be a telepítéskor megadott jelszóval rendszergazdaként a MySQL parancssorába, és listázza ki a adatbázisokat.

```
mysql -u root -p, SHOW DATABASES;
```

♠ **1.14. feladat:** A */root/students.sql* fájlban található exportált adatbázist töltsse fel a MySQL szerverre.

1.15. feladat: SQL parancsok segítségével ismerje meg az egyszerű mintaadatbázist! Találja ki, mi lehet a célja az adatbázisnak, és melyik táblában mit tárol!

```
SHOW DATABASES;, DESCRIBE students.students;
```

♠ **1.16. feladat (opcionális):** Töltsön fel táblánként legalább egy-egy új rekordot az adatbázisba, amelyek megfelelnek a sémáknak, és a táblák összefüggéseinek is. Az új hallgató neve legyen „Mekk Elek” az Ön NEPTUN kódjával. Mekk Elek 1974. április 1-én született.

```
SELECT, INSERT
```

2 — Az Apache 2 webkiszolgáló

Az Apache a Linux rendszereken – és talán a világon – a legszélesebb körben használt webkiszolgáló (lásd 2. és 3. ábra). Az Apache 2 beállítása egyszerű szöveges beállítófájlokban elhelyezett direktívákkal történik. A beállítások a következő fájlokba és könyvtárakba vannak szétosztva `apache1,linuxhalo`:

`apache2.conf` : Az elsődleges Apache 2 konfigurációs fájl. Az Apache 2 globális beállításait tartalmazza.

`conf.d` : Az Apache 2-re globálisan érvényes beállítófájlokat tartalmaz.

`envvars` : Az Apache 2 környezeti változóit tartalmazza.

`httpd.conf` : Történetileg az elsődleges Apache 2 konfigurációs fájl, amelyet a `httpd` démonról neveztek el. Ez a fájl felhasználóspecifikus beállításokat tartalmazhat, amelyek globálisan befolyásolják az Apache 2-t.

`mods-available` : Ez a könyvtár a modulok betöltésére és beállítására szolgáló konfigurációs fájlokat tartalmaz. Nem minden modulhoz egy-egy beállítófájl tartozik.

`mods-enabled` : Szimbolikus linkeket tartalmaz az `/etc/apache2/mods-available` fájljaira. A modul konfigurációs fájljára mutató szimbolikus link létrehozása után az adott modul bekapcsolásra kerül az Apache 2 következő újraindításakor.

`ports.conf` : Az Apache 2 által figyelt TCP portokat tartalmazza. A `Listen` direktíva megadja azt a portot és opcionálisan IP-címet, amelyen az Apache 2-nek figyelnie kell a kéréseket. Ha az IP-cím nincs megadva, akkor az Apache 2 a géphez rendelt minden IP-címen figyelni fog. A `Listen` direktíva alapértelmezett értéke a 80.

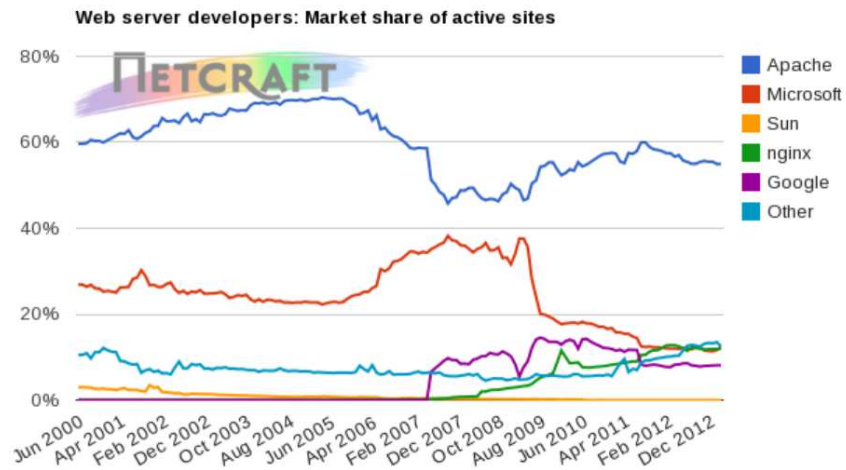
`sites-available` : Ez a könyvtár az Apache 2 virtuális kiszolgálóinak konfigurációs fájljait tartalmazza

`sites-enabled` : A `mods-enabled` mintájára a `sites-enabled` szimbolikus linkeket tartalmaz az `/etc/apache2/sites-available` könyvtárra.

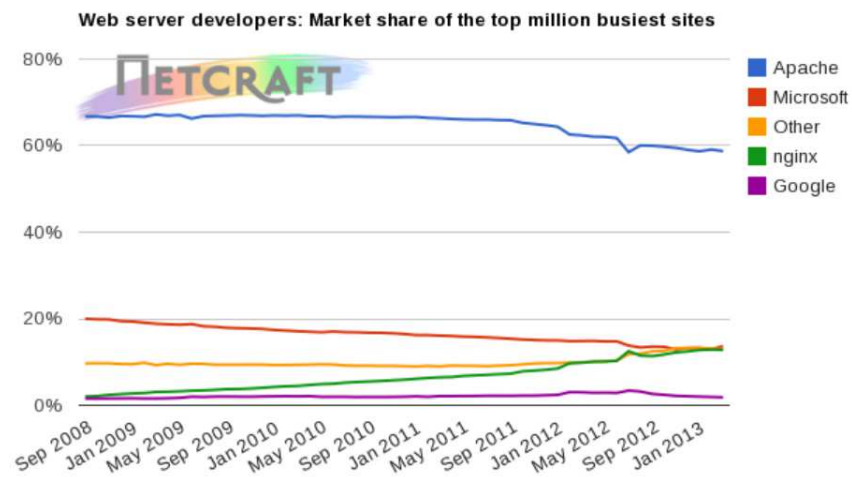
Az Apache 2 induláskor beolvassa a MIME-dokumentumtípusokat tartalmazó fájlt is¹, ennek nevént a `TypesConfig` direktíva adja meg. Ennek alapértelmezett értéke a `/etc/mime.types`.

¹Ez a fájl tartalmazza a „fájlvégződés – tartalom típusa” összerendeléseket.

2. AZ APACHE 2 WEBKISZOLGÁLÓ



2. ábra. A (aktív) HTTP szerverek piaci részesedése a Netcraft mérései alapján



3. ábra. A vezető piaci szereplők HTTP szervereinek piaci részesedése típus szerint a Netcraft adatai alapján

Apache modulok

Az Apache 2 egy moduláris kiszolgáló. Ez azt jelenti, hogy a kiszolgáló maga csak a legalapvetőbb szolgáltatásokat tartalmazza. A bővített szolgáltatások az Apache 2-be tölthető modulokban érhetők el. Ha a kiszolgálót dinamikusan betöltött modulok használatára fordítják, akkor a modulok külön is lefordíthatók és engedélyezhetők, ellenkező esetben az Apache 2-t újra kell fordítani a modulok hozzáadásához vagy eltávolításához [1].

A telepített modulokat a `/etc/apache2/mods-available` könyvtár tartalmazza. Új modulok (esetünkben az MySQL hitelesítés modul) a rendszer csomagkezelőjével telepíthetők:

```
1 $ sudo apt-get install libapache2-mod-auth-mysql
```

Modulok engedélyezése az `a2enmod` segédprogram segítségével lehetséges:

```
1 $ sudo a2enmod auth_mysql
2 $ sudo /etc/init.d/apache2 restart
```

Hasonlóképpen az `a2dismod` segítségével letilthatók az egyes modulok:

```
1 $ sudo a2dismod auth_mysql
2 $ sudo /etc/init.d/apache2 restart
```

Virtuális kiszolgálók

Az Apache 2 egyik leghatékonyabb szolgáltatása, hogy egyetlen gépen több webkiszolgálót is képes futtatni [1]. Ehhez a `VirtualHost` szolgáltatást lehet igénybe venni. Alapértelmeben az Apache 2-ben egyetlen virtuális kiszolgáló van beállítva. Az alapértelmezett virtuális kiszolgáló adatait a

`/etc/apache2/sites-available/default` fájl tartalmazza.

Új virtuális kiszolgáló létrehozásához ezt a fájlt célszerű mintaként felhasználni.

```
1 $ sudo cp /etc/apache2/sites-available/default
2 /etc/apache2/sites-available/irulabor
```

A fájlban található beállítások csak az adott virtuális kiszolgálóra lesznek érvényesek. Ha egy direktíva nincs megadva a virtuális kiszolgálóra, akkor a globális, rendszer-szintű (alapértelmezett) beállítás kerül felhasználásra.

A `DocumentRoot` direktíva megadja, hogy az Apache 2 hol keresse a webhely felépítő fájljait.

A `ServerName` (elhagyható) direktíva megadja, hogy a webhely mely FQDN-re válaszoljon. Az alapértelmezett virtuális kiszolgálóhoz nincs megadva a `ServerName` direktíva, így minden kérésre válaszol, amely nem illeszkedik egy másik virtuális kiszolgálón beállított `ServerName` direktívára.

A következő beállítás hatására például a webhely minden `.irulabor.vmware` végű tartománykérésre válaszolni fog.

2.1. Az Apache biztonsági beállításai 2. AZ APACHE 2 WEBKISZOLGÁLÓ

```
1 ServerName irulabor.vmware
2 ServerAlias *.irulabor.vmware
```

Az `/etc/apache2/sites-available` könyvtárat nem dolgozza fel az Apache 2. Az `/etc/apache2/sites-enabled` alatti szimbolikus linkek mutatnak az elérhető oldalakra.

Új virtuális hosztot az `a2ensite` segédprogram használatával lehet engedélyezni:

```
1 $ sudo a2ensite irulabor
2 $ sudo /etc/init.d/apache2 restart
```

Hasonlóképpen az `a2dissite` segédprogrammal tiltható le egy webhely:

```
1 $ sudo a2dissite irulabor
2 $ sudo /etc/init.d/apache2 restart
```

Az Apache biztonsági beállításai

„Uff, Nagyfőnök! Béla volnék”, avagy az „apacs autentikáció”

A legegyszerűbb, fájl alapú autentikáció engedélyezéséhez a következő sorokkal kell kiegészíteni a virtuális kiszolgáló beállítását:

```
1 AuthType Basic
2 AuthName "Vedett oldal"
3 AuthUserFile /etc/apache2/conf.d/.htpasswd
4 Require valid-user
```

ahol `AuthType` az autentikáció típusát, `AuthName` a hely egy opcionális nevét, a felhasználók neveit és jelszavait tartalmazó fájl elérési útvonalát pedig a `AuthUserFile` direktíva adja meg. A `Require` sorban pedig azt adtuk meg, hogy csak a fájlban szereplő felhasználók jelentkezhetnek be.

Lássunk egy életből elesett példát:

```
1 <Directory /var/www/irulabor/vedett>
2     Options Indexes FollowSymLinks MultiViews
3     AllowOverride None
4     Order allow,deny
5     allow from all
6     AuthType Basic
7     AuthName "Vedett oldal fajl alapu autentikacioval"
8     AuthUserFile /etc/apache2/conf.d/.htpasswd
9     Require valid-user
10 </Directory>
```

A `htpasswd` fájlt (`/etc/apache2/conf.d/.htpasswd` a fenti mintapéldában) a `htpasswd` program segítségével lehet létrehozni és módosítani. Ezt a programot célszerű használni akkor is, amikor egy jelszavat szeretnénk lecserélni.

Egy új bejegyzés a következőképp helyezhető el a fájlban:

```
1 $ htpasswd /etc/apache2/conf.d/.htpasswd ujfelhasznalo
2 New password:
3 Re-type new password:
```

A fenti utasítás hatására a következő bejegyzés kerül a `htpasswd` fájlba:

```
1 ujfelhasznalo:Po9FhxMKQJcRY
```

Végezetül felhasználó törlése a következő sor bemásolásával lehetséges:

```
1 $ htpasswd -D .htpasswd ujfelhasznalo
```

Váltás Apache 2.4-re

Az Apache 2.4 konfigurációja több pontban is eltér a 2.2-es verzióban megszokottól. Az alapértelmezett beállítások megkövetelik, hogy a virtuális kiszolgálók konfigurációs állományainak `conf` legyen a kiterjesztése, de a változtatások érintik a hozzáférés-szabályozást, illetve az autentikáció bizonyos formáit is.

A változtatásokról részletesen a <https://httpd.apache.org/docs/2.4/upgrading.html> oldal ír, azonban az Apache 2.4 esetén is használhatóak a 2.2-es verzióhoz létrehozott konfigurációs állományok, csupán a `mod_access_compat`-ot kell engedélyezni hozzá.

Hozzáférések szabályozása 2.4-ben

Az Apache 2.2-es verziójában a kliens neve, IP címe vagy más karakterisztikája alapján történő hozzáférés szabályozást a `Order`, `Allow`, `Deny` és `Satisfy` direktívák segítségével lehetett megoldani. Ezzel szemben a 2.4-es verzióban a hozzáférés szabályozást az autentikációhoz hasonlóan kell megoldani.

Az alábbi pár példa a különbségeket mutatja:

- minden kérés tiltása

```
1 # Apache 2.2
2 Order deny,allow
3 Deny from all
```

```
1 # Apache 2.4
2 Require all denied
```

- minden kérés engedélyezése

```
1 # Apache 2.2
2 Order allow,deny
3 Allow from all
```

```
1 # Apache 2.4
2 Require all granted
```

- hozzáférés engedélyezése a example.org domén számára, minden más domén tiltása

```
1 # Apache 2.2
2 Order Deny,Allow
3 Deny from all
4 Allow from example.org
```

```
1 # Apache 2.4
2 Require host example.org
```

A .htaccess fájl

A .htaccess (hypertext access) fájl egy könyvtár szintű konfigurációs fájl, amely a webszerver decentralizált menedzsmentjét teszi lehetővé. A webes tartalmak között kerül elhelyezésre, és az adott könyvtár meglátogatásakor lehetővé teszi a szerver némely beállításának felülbírálatát. Engedélyezéséhez a virtuális kiszolgáló számára be kell kapcsolni az *AllowOverride* funkciót.

A *.htaccess* tartalmának módosításával a következő feladatok oldhatók meg:

- egyéni hibalapok,
- jelszavas védelem,
- SSI engedélyezése .htaccess-en keresztül,
- látogatók tiltása IP alapján,
- alapértelmezett fájl megváltoztatása (pl. index.html-ről indulolap.php),
- átirányítások,
- .htaccess tartalom megtekintésének tiltása,
- MIME típusok hozzáadása,
- fájlok direkt linkelésének (más honlapokra) tiltása,
- könyvtárlistázás tiltása,
- stb.

Lássunk két egyszerű példát! Minden szerveren van egy olyan beállítás, mely azt határozza meg, hogy egy könyvtár nevét beírva mely fájlok jelenítődjenek meg a böngészőben. Ez általában az index.html vagy index.htm, index.php szokott lenni, de tetszés szerint módosítható. Ehhez mindössze a .htaccess fájlt kell módosítani:

```
1 DirectoryIndex fajlneve.html
```

Gyakran előfordul, hogy a honlap (vagy egy része) módosul, új helyre költözik. Az átirányítást többféleképpen is meg lehet oldani. Az egyik lehetőség a `.htaccess:`

```
1 Redirect /regikonyvtar/ http://www.honlapodcime.hu/ujkonyvtar/
```

Feladatok

♠ **2.1. feladat:** Telepítse az `apache2` csomagot a beépített csomagkezelőn keresztül!

```
1 $ apt-cache search apache2
2 $ apt-get install apache2
```

2.2. feladat: Ellenőrizze, hogy a webkiszolgáló beállítása lehetővé teszi-e a 80-as porton történő „hallgatózást”. Melyik fájl tartalmazza ezt a beállítást? Milyen paranccsal tudja ezt ellenőrizni böngésző nélkül?

♠ **2.3. feladat:** Az Apache 2 dokumentációja és az alap sablon alapján állítson be egy virtuális kiszolgálót, mely az `irulabor.vmware` domén névre töltődik be. A kiszolgáló által visszaküldött html oldalak kódja megtalálható és letölthető a <https://github.com/ng201/iru> címen. Ne feledje el aktiválni az elkészült konfigurációt!

Az `/etc/hosts` fájlban készítsen el egy bejegyzést, hogy a virtuális gép ismerje a `irulabor.vmware` nevet.

♠ **2.4. feladat:** A fizikai gépen (tehát nem a virtuális linuxon!) töltsse be a böngészőben a `http://irulabor.vmware/vedett` címet. Mint láthatja, a `vedett` mappában található fájlok jelenleg elérhetőek a fizikai gépről is. A feladat az, hogy a `vedett` mappa tartalma csak a virtuális gépről legyen elérhető a webszerverbe épített IP korlátozás segítségével.

♠ **2.5. feladat:** A feladat az, hogy a védett mappa tartalma csak felhasználónév és jelszó segítségével legyen elérhető a webszerverbe épített korlátozás segítségével. A felhasználókat és a hozzájuk tartozó jelszavakat egy külön fájl tartalmazza (`/etc/apache2/conf.d/.htpasswd`). A felhasználók között, természetesen, legyen ott „mekkelek”, akinek legyen a jelszava az Ön NEPTUN kódja!

♠ **2.6. feladat:** A feladat az, hogy a `nagyonvedett` mappa tartalma csak felhasználónév és jelszó segítségével legyen elérhető a webszerverbe épített korlátozás segítségével. A felhasználók a rendszer beépített felhasználói. Kíséreljen meg belépni a fenti címen.

`mod_authnz_external`, `pwauth`

2.7. feladat: A weboldal tartalmaz egy *nyilvanos* elnevezésű mappát is, benne egy *.htaccess* fájlal. Mire alkalmas ez a fájl?

♠ **2.8. feladat:** Az alapértelmezett webkiszolgáló konfigurációban ennek a fájlnek a használata tiltva van. Milyen konfigurációs direktívával tudjuk mégis engedélyezni, és ezt hol kell megadni? A kérdés a *nyilvanos* mappára vonatkozik, csak ott akarjuk engedélyezni a *.htaccess* használatát.

AllowOverride

♠ **2.9. feladat:** A *nyilvanos* mappára szeretné bekapcsolni a webszerver automatikus listázó funkcióját. Mit és milyen formátumban kell ehhez beírni a *.htaccess* fájlba?

+Indexes

3 — Linux héjprogramozás

A Linux héjprogram (shell script) nem más mint (héj)parancsok sorozata, amelyeket az újrafelhasználás jegyében egy fájlba gyűjtünk. A héjprogramok előnye a klasszikus programozási nyelvekkel szemben az, hogy sokkal magasabb, elvontabb szintem lehet bennük dolgozni, és megkönnyítik a fájlokkal és könyvtárakkal való manipulációt. A futási időben történő interpreter-használat miatt a sebességük elmaradhat a klasszikus nyelvekéhez képest, de a gyorsabb fejlesztési ciklus oltárán ez gyakran feláldozható [3].

Első lépések

Kezdjük egy egyszerű programmal. Határozzuk meg, hogy hány felhasználó van aktuálisan bejelentkezve egy többfelhasználós Linux szerverre. Ebben a feladatban a `who` parancs lehet a segítségünkre:

```
1 $ who
2 frakk pts/2 Dec 31 16:39 (magyarvizsla.pelda.hu)
3 lukrecia pts/3 Dec 27 11:07 (cicavizio.pelda.hu)
4 szerenke dtlocal Dec 27 17:55 (macska.pelda.hu)
5 karolybacsi pts/5 Dec 27 17:55 (:32)
6 irmaneni pts/14 Jan 2 06:42
```

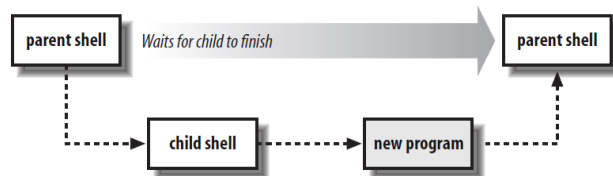
Egy,kettő, három, négy, öt... Egy nagy, többfelhasználós rendszerben azonban a lista hosszúra nyúlhat, így a sorok megszámlálása nehézkessé válhat. Itt a lehetőséget a feladat automatizálására.

A bejelentkezett felhasználók megszámlálásához módosítsunk egy picit a programon, használjuk fel a `wc` programot, amely megszámlolja a bemenetén a sorokat, szavakat és karaktereket. A két program „összekapcsolására” használjuk a `/-`-t:

```
1 $ who | wc -l
2 5
```

A következő lépés az előző utasítás páros egy parancssá tétele. Ehhez az előző programot egy fájlba kell elhelyezni, majd a fájlt futtathatóvá kell tenni:

```
1 $ cat > nusers
2 who | wc -l
3 ^D
4 $ chmod +x nusers
5 $ ./nusers
6 5
```



4. ábra. Héjprogramok futtatása [3]

#!

Amikor az előző programot futtatjuk, akkor a Linux rendszermag egy új folyamatot indít, ezen folyamaton belül próbálja meg futtatni a programot. A rendszermag hagyományos (lefordított) programok esetében tudja, hogyan lehet ezt megtenni, de a héjprogramok esetében ez nincs így. Ilyenkor a hívó oldal feltételezi, hogy a program egy héjprogram, és elindítja a `/bin/sh` értelmezőt, hogy futtassa le az héjprogramot (a mechanizmust a 4. ábra szemlélteti).

Ez a mechanizmus jól működik, amíg csak egy shell van telepítve a rendszerben. Ellenkező esetben a héjprogram első sora nyújt(hat) segítséget: ha az első sora a `#!` jelekkel kezdődik, amelyeket szorosan követ az értelmező neve, akkor azzal az értelmezővel futtatja le a héjprogramot:

```
1 #!/bin/bash
2 #
3
4 who | wc -l
```

Az `echo` parancs

Az `echo` parancs feladata változó értékének, illetve szövegek, karakterfüzerek kiírása a képernyőre. Az `echo` parancs kiírja az argumentumait a képernyőre, majd pedig egy új sort kezd:

```
1 $ echo Hol volt, hol nem volt,
2 Hol volt, hol nem volt,
3 $ echo "volt egyszer egy szép kis kertés ház."
4 volt egyszer egy szép kis kertés ház.
5 $ echo Abban lakott két kövér macska.
6 Abban lakott két kövér macska.
```

Ha az `echo` első argumentuma `-n`, akkor a sorvégi újsor karaktert nem „írja” ki (az alulvonás a kurzor pozícióját jelöli):

```
1 $ echo -n "Adja meg a nevet: "
2 Adja meg a nevet: _
```

Az `-e` kapcsoló segítségével lehet az `echo`-t rábírní a szóktetett karakterek értelmezésére:


```

1 $ echo "\n\n\n\n"
2 \n\n\n\n
3 $echo -e "Frakk\nSzerenke\nLukrecia"
4 Frakk
5 Szerenke
6 Lukrecia

```

A cut

A `cut` segítségével előre megadott karakterek mentén oszlopokra bonthatjuk a be-meneti állományt.

```

1 $ echo -e "1:23:456:A:B:C:D\n2:46:912:AA:BB:CC:DD" | cut -d":" -f2,4-6
2 23 A B C
3 46 AA BB CC

```

A date parancs

A `date` parancs segítségével kiírathatjuk az aktuális dátumot és időt különböző formátumban. Így például a

```

1 $ date +"%Y. %m. %d."

```

parancs hatására a dátum klasszikus magyar formátumra emlékeztető formátumban kerül kiírásra. Ha például az aktuális hónap nevére vagyunk kíváncsiak, akkor a `date` parancs formátumleírásában a `%b` vagy a `%B` opciók segíthetnek:

```

1 $ date +"%b"
2 Jan
3 $ date +"%B"
4 January

```

A bash

A `bash` egy Unix/Linux rendszerhéj/shell. Az elnevezés egy játékos mozaikszó, amely a *Bourne again shell*, illetve a *born again shell* kifejezéseket rövidíti.

Változók

A `bash`-ban is használhatóak változók, mint bármely más programozási nyelvben. Nincsenek típusok, a változók számokat, karaktereket és stringeket is tárolhatnak. Változók deklarálására nincs szükség, értékadással lehet őket létrehozni.

```

1 $ envváltozom=Frakk_a_macskak_reme
2 $ echo $envváltozom
3 Frakk_a_macskak_reme

```

Értékadásnál figyelni kell, hogy a változó nevét közvetlenül követi az egyenlőségjel, majd pedig az érték. Ha az értékben üres karakterek is vannak, akkor aposztrófot kell használni az érték elején és végén:

```
1 kutya=Frakk macska1=Szerenke macska2=Lulkrecia
2 # ^^ ertekadas tobb valtozonak egy sorban
3 bacsi="Karloly bacsi"
4 # ^^ ureskarakterek eseten aposztrofot kell hasznalni
```

Változók értékeinek a konkatenációja is gyerekjáték:

```
1 allatok="$kutya, $macska1, $macska2"
```

További példákat változók értékadására az alábbi héjprogramban láthatunk:

```
1 #!/bin/bash
2
3 a=375
4 hello=$a
5
6 echo $hello          # 375
7 # hivatkozás a változóra
8
9 echo ${hello}       # 375
10 # mint előbb
11
12 # idézőjelek...
13 echo "$hello"      # 375
14 echo "${hello}"    # 375
15 echo '$hello'      # $hello
16 # a változó behelyettesítést letiltotta az aposztróf
17
18 u=`who | wc -l`
19 # ^^^ a parancs futási eredménye a változóba kerül
20
21 u=$(( who | wc -l ))
22 # ^^^ a parancs futási eredménye a változóba kerül
```

Aritmetika *bash*-ban

Bash-ban az aritmetikai kifejezések kiértékelésére a `$((kifejezes))` kód használható. A kifejezés kiértékelése egész szám aritmetikával történik, túlcsoordulás ellenőrzése nélkül. Az operátorok és rájuk érvényes szabályok a C nyelv szabályait követik.

```
1 $ x=100
2 $ y=3
3 $ echo $((x/y))
4 33
5 $ echo $((y/x))
6 0
7 $ echo $((++x))
8 101
```

Parancsok futtatási eredményének tárolása változóban

Parancsok futási eredményét a `$(...)` vagy `'...'` segítségével lehet egy változóban eltárolni:

```
1 x=`echo "12" `
2 y=$(echo "almafa")
3 echo x # 12
4 echo y # almafa
5 OUTPUT=$(ls -l) # több sor van a kimenetben
6 echo "${OUTPUT}"
```

Elágazások

Az elágazások *bash* formája a következő:

```
1 if [expression];
2 then
3
4 kód, ha a kifejezés igaz
5
6 fi
```

Számok összehasonlítása I. Írjunk egy egyszerű számológépet *bash*-ban.

```
1 #!/bin/bash
2
3 inp1=12
4 inp2=11
5 echo "1. Összeadás"
6 echo "2. Kivonás"
7 echo "3. Szorzás"
8 echo -n "Válasszon egy aritmetikai műveletet [1,2 vagy 3]? "
9 read oper
10
11 if [ $oper -eq 1 ]
12 then
13     echo "Az összeadás eredménye: " $(( $inp1 + $inp2 ))
14 else
15     if [ $oper -eq 2 ]
16     then
17         echo "A kivonás eredménye: " $(( $inp1 - $inp2 ))
18     else
19         if [ $oper -eq 3 ]
20         then
21             echo "A szorzás eredménye: " $(( $inp1 * $inp2 ))
22         else
23             echo "Nem beszélni magyar?"
24         fi
25     fi
26 fi
```

Számok összehasonlítása II. Írjunk egy egyszerű programot, amely két beolvasott számról eldönti, hogy melyik a nagyobb.

```
1 #!/bin/bash
2 #!/bin/bash
3
4 echo "1. szám: "
5 read a
6 echo "2. szám: "
7 read b
8
9 if [ "$a" -eq "$b" ];
10 then
11     echo "a két szám egyforma";
12 else
13     if [ "$a" -lt "$b" ];
14     then
15         echo "az első szám a kisebb"
16     else
17         echo "az első szám a nagyobb "
18     fi
19 fi
20
21 ## -eq # equal
22 ## -ne # not equal
23 ## -lt # less than
24 ## -le # less than or equal
25 ## -gt # greater than
26 ## -ge # greater than or equal
```

Szövegek összehasonlítása Írjunk egy bash scriptet, amely kiírja, hogy hány napos az aktuális hónap.

```
1 #!/bin/bash
2
3 month=$(date +%b)
4 # echo "$month"
5
6 if [ $month == "Jan" ] || [ $month == "Mar" ] ||
7   [ $month == "May" ] || [ $month == "Jul" ] ||
8   [ $month == "Aug" ] || [ $month == "Oct" ] ||
9   [ $month == "Dec" ]
10 then
11     echo "31"
12 else
13     if [ $month != "Feb" ]
14     then
15         echo "30"
16     else
17         echo "?????"
18     fi
19 fi
```

A fenti kódrészletben az egyetlen elvarratlan dolog a február, amelynek a hossza függ az évszámtól, azaz, hogy az adott év éppen szökőév-e. Szökőévek a következők: minden négyvel osztható év, kivéve a százzal is oszthatókat. Szökőévek viszont a 400-zal osztható évek. A definíció alapján a szökőéveket az alábbi sorokkal tudjuk meghatározni. Az oszthatóság tesztelésére egy segédprogramot, a *bc*-t használhatjuk:

```

1 year=$( date +%Y )
2 sz=$((year%4))
3
4 if [ $sz -ne 0 ]
5 then
6     echo "28"
7 else
8     sz=$((year%400))
9
10    if [ $sz -eq 0 ]
11    then
12        echo "29"
13    else
14        sz=$((year%100))
15        if [ $sz -eq 0 ]
16        then
17            echo "28"
18        else
19            echo "29"
20        fi
21    fi
22 fi

```

Ciklusok

A *bash* *for* ciklus jelentősen különbözik a C/C++-ban megismert *for* ciklustól:

```

1 for arg in [list]
2 do
3     utasítás(ok)...
4 done

```

Nézzük meg, milyen módon tudnánk kiíratni a Naprendszerben található bolygók neveit:

```

1 #!/bin/bash
2 # Bolygok listazasa.
3 for planet in Merkur Venusz Fold Mars Jupiter Szaturnusz
4             Uranusz Neptunusz Pluto
5 do
6     echo $planet # soronkent egy-egy bolygó
7 done
8 echo; echo "Fene! A Pluto mar nem bolygo!"
9 exit 0

```

Parancssori argumentumok

A scripteket tetszőleges számú parancssori argumentummal meghívhatjuk. Ilyenkor a $\$1, \$1, \dots, \$9, \${10}, \dots$ változókkal hivatkozhatjuk meg őket. A $\$*$ változó az összes parancssori paramétert tartalmazza, míg a $\$#$ változó a parancssori argumentumok számát adja meg.

```
1 #!/bin/bash
2
3 # Hívja meg a scriptet legalább 10 paraméterrel, például
4 # ./scriptname 1 2 3 4 5 6 7 8 9 10
5
6 MINPARAMS=10
7
8 echo "A script neve \"\$0\"."
9 # elérési útvonala, az aktuális könyvtár esetében ez a ./
10
11 echo A script neve \"`basename $0`\".
12 # kiszűri az elérési útvonalat
13
14
15 echo "A parancssori paraméterek rendre: \"$*"
16 if [ $# -lt "$MINPARAMS" ]
17 then
18     echo "A scriptet legalább $MINPARAMS parancssori"
19     echo "argumentummal kell meghívni!"
20     exit 1
21 fi
22
23 if [ -n "$1" ] # változó értékének a tesztelése
24 then
25     echo "Az #1. parameter erteke $1"
26 fi
27 if [ -n "$2" ]
28 then
29     echo "A #2. parameter erteke $2"
30 fi
31 if [ -n "$3" ]
32 then
33     echo "A #3. parameter erteke $3"
34 fi
35
36 # ...
37
38 if [ -n "${10}" ] # A 9-nél nagyobb sorszámú paramétereket
39                 # {zárojelekbe} kell tenni.
40 then
41     echo "A #10. parameter erteke ${10}"
42 fi
43
44 exit 0
```

Függvények *bash*-ban

A többi programozási nyelvhez hasonlóan a *bash*-ban is hozhatunk létre függvényeket, habár első pillantásra furának tűnhetnek. Felépítésük a következő sémát követik:

```
1 function_name () {
2
3 utasítás(ok)...
4
5 }
```

Egyszerű függvényre az alább látható példa:

```
1 #!/bin/bash
2
3 JUST_A_SECOND=1
4
5 # A deklarációnak meg kell előznie a használatot
6 fun ()
7 {
8     i=0
9     REPEATS=30
10
11     sleep $JUST_A_SECOND # Hé, várjunk csak egy pillanatot!
12     while [ $i -lt $REPEATS ]
13     do
14         echo "-----FUNCTIONS----->"
15         echo "<-----ARE-----"
16         echo "<-----FUN----->"
17         echo
18         let "i+=1"
19     done
20 }
21
22 # Függvény meghívása.
23
24 fun
25
26 exit $?
```

A függvényt meg lehet hívni argumentumokkal is.

```
1
2 #!/bin/bash
3 # Functions and parameters
4
5 DEFAULT=default # Default param value.
6
7 func2 () {
8     if [ -z "$1" ] # Is parameter #1 zero length?
9     then
10         echo "-Parameter #1 is zero length.-" # Or no parameter passed.
11     else
12         echo "-Parameter #1 is \"$1\".-"
13     fi
```

```
14
15 variable=${1-$DEFAULT}
16                                     # What does
17 echo "variable = $variable" #+ parameter substitution show?
18                                     # -----
19                                     # It distinguishes between
20                                     #+ no param and a null param.
21
22 if [ "$2" ]
23 then
24     echo "-Parameter #2 is \"$2\".-"
25 fi
26
27 return 0
28 }
29
30 echo
31
32 echo "paraméter nélkül"
33 func2
34
35 ## *** kimenet *** ##
36 #-Parameter #1 is zero length.-
37 #variable = default
38
39 echo "nulla hosszú paraméterrel"
40 func2 ""
41
42 ## *** kimenet *** ##
43 #-Parameter #1 is zero length.-
44 #variable =
45
46 echo "null paraméterrel"
47 func2 "$uninitialized_param"
48
49 ## *** kimenet *** ##
50 #-Parameter #1 is zero length.-
51 #variable =
52
53 echo "egy paraméterrel"
54 func2 first
55
56 ## *** kimenet *** ##
57 #-Parameter #1 is "first".-
58 #variable = first
59
60 echo "két paraméterrel"
61 func2 first second # Called with two params
62
63 ## *** kimenet *** ##
64 #-Parameter #1 is "first".-
65 #variable = first
66 #-Parameter #2 is "second".-
67
```



```

68 echo "\"\" és \"second\" paraméterekkel"
69 func2 "" second
70
71 ## *** kimenet *** ##
72 -Parameter #1 is zero length.-
73 variable =
74 -Parameter #2 is "second".-
75
76 exit 0

```

Palindromák keresése *bash*-ban

Palindromák, azaz olyan szavak, amelyek előlről és hátulról olvasva is ugyanazt adják, detektálása például a *tac* program segítségével oldható meg. A *tac* a *cat* fordítottja.

```

1 #!/bin/bash
2
3 function ispalin { [ "$( echo -n $1 | tac -rs . )" = "$1" ]; }
4
5 echo "$(ispalin god && echo yes || echo no) "
6 echo "$(ispalin lol && echo yes || echo no) "

```

Karakterfolyamok kezelése

Reguláris kifejezések: BRE és ERE

A reguláris kifejezések olyan struktúrák, amelyek lehetővé teszik olyan karakterfüzéreket, szövegrészek keresését, amelyek megfelelnek egy keresési feltételnek (pl. „mássalhangzóval kezdődik”, stb).

A Linux és Unix rendszerek programjainak nagy része kihasználja a reguláris kifejezések erejét (lásd 2. ábra). Így például

- a *grep* és családja (*grep*, *egrep*, *agrep*), amely lehetővé teszi egy adott mintára illeszkedő sorok kiválogatását egy szövegből,
- a *sed*, amely lehetővé teszi a bemeneti folyamat megváltoztatását,
- a karakterfüzéreket feldolgozó nyelvek, mint például az *awk*, a *Perl*, a *Ruby*, a *Tcl*,
- a fájlkezelők, mint például a *more*, *page* és a *pg* és a *less*,
- a fájl szerkesztők, mint például az *ed*, a *vi*, az *emacs*, a *vim*, stb.

A reguláris kifejezéseknek két típusa van használatban [3], a *POSIX BRE* (Basic Regular Expression) és a *ERE* (Extended Regular Expression), amelyek a sok-sok különbségük mellett abban az egyben megegyeznek, hogy hagyományos karakterekből és speciális (jelentéssel bíró) karakterekből építkeznek. A két típus karaktereit, szerkezeteit a 1. táblázat mutatja be.

| Karakter | BRE / ERE | Jelentése |
|----------|-----------|---|
| \ | mindkettő | általában a karakter speciális jelentését kapcsolja ki („szókteti”) |
| . | mindkettő | pontosan egy (de az lehet bármilyen) karakterre illeszkedik |
| * | mindkettő | nulla vagy több olyan karakterre illeszkedik, ami az őt megelőző mintával leírható (például a <code>.*</code> bárhány bármilyen karakterre illeszkedik) |
| ^ | mindkettő | sor eleji illeszkedés, azaz az őt követő reguláris kifejezésnek a sor vagy a karakterfüzér elején kell lennie |
| \$ | mindkettő | sor végi illeszkedés, azaz az őt megelőző reguláris kifejezésnek a sor vagy a karakterfüzér végén kell lennie |
| [...] | mindkettő | bármelyik a zárójelekbe zárt karakterre illeszkedik, ahol az esetleges „-” karakterrel intervallumot lehet jelölni |
| \{n,m\} | BRE | legalább n , de maximum m olyan kifejezésre illeszkedik, amely az őt megelőző reguláris kifejezésre illeszkedik |
| \(\) | BRE | megjegyzi a zárójelek közé zárt kifejezésre illeszkedő karakterfüzért, amelyekre később a <code>\1 - \9</code> kifejezésekkel lehet hivatkozni |
| \n | BRE | visszaadja az n -edik ($n = 1, \dots, 9$) zárójelbe zárt mintára illeszkedő karakterfüzért |
| {n,m} | ERE | éppen úgy viselkedik, mint az BRE <code>\{n,m\}</code> |
| + | ERE | egy vagy több olyan karakterre illeszkedik, ami az őt megelőző mintával leírható |
| ? | ERE | nulla vagy egy olyan karakterre illeszkedik, ami az őt megelőző mintával leírható |
| / | ERE | az előtte vagy utána található reguláris kifejezésre illeszkedik (egyszerű „vagy” funkció) |
| () | ERE | a zárójelekbe zárt reguláris kifejezésre illeszkedik |

1. táblázat. Reguláris kifejezések [3]

Lássunk pár egyszerű példát reguláris kifejezésekre:

`tolsztoj` a „tolsztoj” karakterfüzérre illeszkedik bárhol a sorban,

`^tolsztoj` a sor eleji „tolsztoj” füzérre illeszkedik,

`tolsztoj$` a sor végi „tolsztoj” füzérre illeszkedik,

`^tolsztoj$` azokra a sorokra illeszkedik, amelyek pontosan megegyeznek a „tolsztoj” füzérrel,

`[Tt]olsztoj` a „Tolsztoj” és a „tolsztoj” kifejezésre is illeszkedik,

`tols.toj` a „tols”, majd egy tetszőleges karakter, majd „toj” kifejezésre illeszkedik bárhol a sorban,

`tol.*toj` olyan sorokra illeszkedik, amelyekben a „tol” füzért bárhány (akár 0) karaktereket nulla tetszőleges karakter, majd pedig a „toj” füzért követ (pl. `toltoj`, `tolKIAFENEttoj`, stb.).

| típus | grep | sed | ed | ex/vi | more | egrep | awk | lex |
|-------|------|-----|----|-------|------|-------|-----|-----|
| BRE | ✓ | ✓ | ✓ | ✓ | ✓ | | | |
| ERE | | | | | | ✓ | ✓ | ✓ |

2. táblázat. Linux programok és az általuk használt reguláris kifejezések típusa [3]

Fontos kiemelnünk a BRE-ek „visszaemlékező képességét”, azaz azt a tulajdonságát, hogy egy korábbi illeszkedő mintára illeszkedő karakterfüzérre lehet később hivatkozni. Ezt a funkcionalitást két lépésben lehet elérni. Elsőként egy alkifejezést zárójelek, `\(` és `\)`, közé kell zárni. Ezt követően a `\1` - `\9` kifejezésekkel lehet hivatkozni a zárójelekbe zárt reguláris kifejezésekre illeszkedő karakterfüzérekre. Például a `\([\"']\).*\1` reguláris kifejezés minden idézőjelek közé zárt szóra illeszkedik (pl. 'kilincskerék' vagy "szendvics"), sőt, a kifejezés elején és a végén szigorúan megköveteli ugyanazon idézőjel (' vagy ") használatát.

A *grep*

A *grep* (a név egy *ed* parancs, a *g/re/p* – globálisan illeszd és írd ki, rövidítése) egy parancssori program, amely fájlban vagy karakterfolyamban tesz lehetővé karakterfüzerek (sztringek) keresését reguláris kifejezések alapján [4, ?].

A *grep* legegyszerűbben az alábbi módon hívható meg:

```
1 $ grep 'STRING' filename
```

Egyszerű keresés : A

```
ps -ax | grep gnome
```

parancs hatására a kimeneten azok a futó folyamatok jelennek meg, amelyek nevében szerepel a *gnome* kifejezés.

Blob : Gyakran megesik, hogy egy adott mintát a könyvtár összes összes fájlban keresünk. Ezt a következőképp tehetjük meg:

```
grep 'ez egy teszt' *.pl
```

A parancs kiadásának folyamánként a *grep* végigbogarássza az aktuális könyvtárban található összes *.pl* végződésű fájlt az „ez egy teszt” minta után kutatva. A válasz azon fájlok nevét tartalmazza, amelyekben fellelhető a keresett kifejezés.

Találatok helye : A *grep*et a *-n* kapcsolóval meghívva nemcsak a mintára illeszkedő sorokat adja, hanem azok sorszámát is:

```
grep -n gnome processzek.txt
```

Találatok száma : Azt, hogy hányszor találta meg a keresett kifejezést a *grep* a megadott fájlban/karakterfolyamban a *-c* kapcsoló segítségével tudhatjuk meg:

```
grep -c /usr process.txt
```

Nagy- és kisbetű : A *grep* megkülönbözteti a nagy- és kisbetűket. Ez a tulajdonság az *-i* kapcsolóval kikapcsolható

```
ps -ax | grep GnOmE process.txt
ps -ax | grep -i GnOmE process.txt
```

Negálás : Végezetül ha arra lennénk kíváncsiak, hogy mely sorok nem illeszkednek az adott kifejezésre, akkor a *-v* kapcsoló használata a javallot:

```
grep -v /usr process.txt
```

A *sed*

A *sed* (stream editor) tulajdonképpen egy programozható szövegszerkesztő, ami a szabványos bemenetére érkező szöveget képes átalakítani [5]. Működésének lényege, hogy a feldolgozandó szöveget soronként egy átmeneti tárhoz, az úgynevezett mintatérbe olvassa be, szabályos kifejezések alapján megkeres benne bizonyos részeket, majd elvégzi rajtuk az egybetűs parancsok formájában megadott műveleteket (lásd 5. ábra).

A *sed* műveletei a következők:

| | |
|------------------------------|------------------------------|
| <i>p</i> kiíratás, | <i>d</i> törlés, |
| <i>s</i> helyettesítés, | <i>a</i> hozzáfűzés, |
| <i>i</i> beszúrás, | <i>c</i> a mintatér cseréje, |
| <i>y</i> karakterek cseréje. | |

A *sedet* leggyakrabban egy másik programtól érkező kimenet feldolgozására használjuk:

```
1 ... | sed 'sed utasitasok' | ...
```

A *sed* programok egy sora a következőképpen néz ki:

```
<cím1>, <cím2> parancs
```

Itt *<cím1>* és *<cím2>* egy-egy szám vagy reguláris kifejezés lehet. Ha számot adunk meg, az a bemenet adott sorszámú sorának feldolgozását jelenti, ha reguláris kifejezést (ezt két / – perjel – közé kell zárni), akkor a program minden olyan sorra lefut, amelyre a kifejezés illeszkedik. Ennek megfelelően a *25 egybetűs-parancs* utasítás csak a bemenet 25. sorát fogja érinteni, míg a */[0-9]/ egybetűs-parancs* programsor minden olyan szövegsort érinteni fog, amelyben legalább egy számjegy van.

Egyszerű példák:

1. a fájl első tíz sorának a törlése:

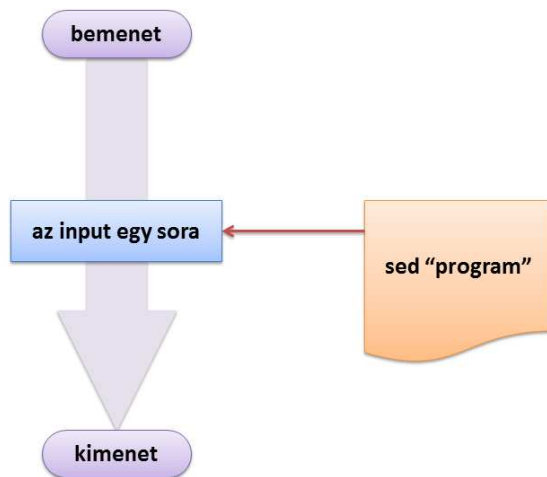
```
sed -e '1,10d' /etc/services
```

2. a *//*-rel kezdődő, megjegyzéseket tartalmazó sorok törlése egy C++ forrásfájlból:

```
cat kilincskerek.cpp | sed '/^\/\//d'
```

3. a bemenetben a Linux kifejezés lecserélése Linux-Unix-ra:

```
echo "szeretem a Linuxot" | sed 's/Linux/Linux-Unix/'
```

5. ábra. A *sed* működési elve

4. a Linux kiszolgáló IP címének a meghatározás:

```
ifconfig eth0 | grep 'inet addr:' |
sed 's/^.+inet addr:([0-9\.]*).*$/\1/'
```

5. utolsó három karakter törlése a fájl összes sorának a végéről:

```
sed 's/...$//' bemenet.txt
```

És egy egyszerű oszlopcseré a'la *sed*:

```
1 $ cat test1
2 first:second
3 one:two
4 $ sed 's/\(.*\):\(.*\)/\2:\1/' test1
5 second:first
6 two:one
```

Végezetül írjuk ki az aktuális dátumot magyar formátumban, azaz a hónapok legyenek római betűkkel írva:

```
1 $ date +"%Y. %m. %d." | sed 's/\. 01\./\ . I.\./'
2 | sed 's/\. 02\./\ . II.\./'
3 | sed 's/\. 03\./\ . III.\./'
4 | sed 's/\. 04\./\ . IV.\./'
5 | sed 's/\. 05\./\ . V.\./'
6 | sed 's/\. 06\./\ . VI.\./'
```

Az *awk*

Az *awk* egy általános célú programozási nyelv, amelyet szöveges állományok feldolgozására terveztek. Elnevezése a megalkotói – Alfred Aho, Peter Weinberger és Brian Kernighan – családnevének kezdőiből született [5, 3].

Az *awk*-ot a következőképp lehet meghívni a parancssorból:

```
1 $ awk [ -F fs ] [ -v var=value ... ] 'program' [ -- ]
2 [ var=value ... ] [ file(s) ]
```

vagy pedig a következő formában:

```
1 $ awk [ -F fs ] [ -v var=value ... ] -f programfile [ -- ]
2 [ var=value ... ] [ file(s) ]
```

Rövid programokat tipikusan a parancssorban szokás megadni, míg hosszabb kódokat külön fájlban lehet megadni a *-f* kapcsoló segítségével. Ráadásul ezt az opciót többször egymás után is meg lehet adni, ilyenkor a végrehajtandó program a megadott programok konkatenációja lesz. Ha nincs fájlnev megadva, akkor az *awk* a programok a standard bemenetről olvassa. A *--* kapcsoló különleges, és azt jelzi, hogy nincs több parancssori paraméter megadva az *awk* számára. Minden ezt követő opció a program számára lesz elérhető.

Az *awk* a bemenetet mint rekordok összességét látja, amelyek mindegyike mezőkre osztható. Alapesetben a rekordok sorok, a mezők pedig egy vagy több nem üres karakterből álló karakterfüzér (stringek). Habár ezeket a beállításokat az *awk* programozó bármikor tetszés szerint megváltoztathatja [3].

Egy tipikus *awk* program a végrehajtása során a bemeneti adatokat egy másféle kimenetű formálja át. A programok általában mintából és a mintához tartozó parancsokból állnak:

```
1 /1. minta / { parancs(ok) }
2 /2. minta / { parancs(ok) }
```

Az *awk* soronként olvassa a bemenetet. Minden beolvasott sort összehasonlít a mintákkal, és ha illeszkedést talál, a parancsokat végrehajtja. A mintákat a reguláris kifejezések szabályai szerint értelmezi.

A *awk* különleges parancsformái a következők:

BEGIN parancs(ok) adatbeolvasás előtt ezeket a parancsokat végrehajtja,

END parancs(ok) adatbeolvasás és a többi parancs végrehajtása után ezeket a parancsokat végrehajtja,

/minta/ ha nincs külön parancs megadva, a mintának megfelelő sort kinyomtatja,

akció ha nincs minta megadva, a parancsokat végrehajtja minden sorra egymás után.

A „Helló, világ!” program *awk*-ban a következőképpen implementálható:

```
1 BEGIN { print "Hello, vilag!" }
```

Az *awk* speciális beépített változói a

FS mező szeparátor (reguláris kifejezés) (alapértelmezett érték: " "),

NF mezők száma a beolvasott sorban,

NR olvasott rekordok száma,

OFS output mező szeparátor (alapértelmezett érték: " "),

ORS output rekord szeparátor (alapértelmezett érték: "\n"),

FILENAME aktuális input fájl neve,

ARGC paraméterek száma,

ARGV paraméterek értéke,

FNR olvasott rekordok száma az aktuális fájlban,

OFMT output számformátum,

RS input rekord szeparátor (alapértelmezett érték: "\n").

Az alábbi *awk* program megszámlolja, majd kiírja a bemeneti fájlban lévő sorok, szavak és karakterek számát úgy, mint a *wc* nevű program:

```
1 {
2   w += NF
3   c += length + 1
4 }
5 END { print NR, w, c }
```

A következő *awk* program a szavak gyakoriságáról készít statisztikát asszociatív tömb felhasználásával:

```
1 BEGIN { RS="^[a-zA-Z]+" }
2
3 { words[tolower($0)]++ }
4
5 END { for (i in words)
6   print i, words[i]
7 }
```

A alábbi *awk* program kiírja a „bemenet.txt” fájl azon sorait, amelyek „Rent”-tel kezdődnek:

```
1 $ awk '/Rent/{print}' file
```

A *print* utasítás elhagyható, mert (ahogy azt már korábban említettük) ez az *awk* alapértelmezett viselkedése:

```
1 $ awk '/Rent/' file
```

A bemeneti fájlban található tábla n -edik oszlopában (az oszlopokat ” ” választja el) levő számok összege „awkul” a következőképp hangzik:

```
1 $ awk -v COLUMN=n '{ sum += $COLUMN } END { print sum }' file
```

Egy apró kis változtatással megoldható, hogy az összeg helyett az n -edik oszlop számainak az átlaga kerüljön a kimenetre. Íme:

```
1 $ awk -v COLUMN=n \textbackslash\textbackslash
2 '{ sum += $COLUMN } END { print sum / NR }' file
```

Végezetül a következő programocska a „lista.txt” fájlban megszámolja, hányszor lelhető fel a „foo” minta:

```
1 $ awk '
2 > BEGIN { print "Analysis of \"foo\"" }
3 > /foo/ { ++n }
4 > END { print "\"foo\" appears " n " times." }' lista.txt
```

A patch

Amikor egy biztonsági frissítés válik elérhetővé egy programhoz, akkor általában azt az *apt-get* segítségével frissítjük.

Előfordulhat azonban, hogy a programot saját magunk fordítottuk forráskódból. Ezekben az esetekben az előző módszer a biztonsági frissítések telepítésére nem nyújt megoldást. Ilyen esetekben a *patch* nyújthat megoldást, amelyet előállítja az eredeti forráskód módosított, javított verzióját. Az így kapott változatot újrafordítva a javított program már könnyen elkészíthető.

A patch fájl egy szöveges fájl, ami egy fájl két verziója közötti különbségeket tartalmazza. Patch fájlokat a *diff* parancs segítségével lehet készíteni.

Patch készítése

A patch készítésének a megértéséhez készítsünk egy apró C nyelven írt programot *hello.c* néven:

```
#include <stdio.h>

int main() {
    printf("Hello World\n");
}
```

Készítsünk egy másolatot a *hello.c* programról *hello_new.c* néven, majd hajtsunk végre pár apró változtatást a kódban:

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello World\n");
    return 0;
}
```


Végezetül készítsük el a patch-ot a *diff* parancs segítségével:

```

1 $ diff -u hello.c hello_new.c > hello.patch
2 The above command will create a patch file named ``hello.patch``.
3
4 --- hello.c 2014-10-07 18:17:49.000000000 +0530
5 +++ hello_new.c 2014-10-07 18:17:54.000000000 +0530
6 @@ -1,5 +1,6 @@
7  #include <stdio.h>
8
9 -int main() {
10 +int main(int argc, char *argv[]) {
11     printf("Hello World\n");
12 + return 0;
13 }
```



6. ábra. Ez egy patch

Patch felhasználása

A *patch* parancs egy patch fájlt vár bemenetként, amely tartalma alapján létrehozza az eredeti fájl (vagy fájlok) a patch-elt verzióját:

```

1 patch -p[num] < patchfile
2 patch [options] originalfile patchfile
```

A *hello.patch* fájl felhasználására mutat példát az alábbi kódrészlet. A parancs futtatása után a *hello.c* fájl tartalma a megváltozott forrásfájllal fog meg egyezni. Figyeljük meg, hogy a patch fájl tartalmazza annak a fájlnek a nevét is, amelyet meg kell változtatnia.

```

1 $ patch < hello.patch
2 patching file hello.c
```

Backup fájl automatikus készítése a patch felhasználása előtt

A patch felhasználása előtt automatikus backup fájl készíthető a *-b* kapcsoló segítségével.

```
1 $ patch -b < hello.patch
2 patching file hello.c
```

A parancs futtatásának eredményeként létrejön egy *hello.c.orig* nevű fájl is, amely az eredeti fájl tartalmát tartalmazza.

A *-v* kapcsoló segítségével a backup fájl nevének a formátuma is megadható.

```
1 $ patch -b -V numbered < hello.patch
2 patching file hello.c
```

Most a backup fájl neve *hello.c. 1* lesz.

Dry-run

Patch fájl ellenőrzésére a *--dry-run* kapcsoló használható.

```
1 $ patch --dry-run < hello.patch
2 patching file hello.c
```

A *--dry-run* kapcsoló esetében az eredeti *hello.c* fájl nem változik meg!

Patch visszavonása

A *-R* kapcsoló segítségével lehetőség van visszavonni egy patch-ot:

```
1 $ patch < hello.patch
2 patching file hello.c
3
4 $ ls -l hello.c
5 -rw-r--r-- 1 lakshmanan users 94 2014-10-07 20:05 hello.c
6
7 $ patch -R < hello.patch
8 patching file hello.c
9
10 $ ls -l hello.c
11 -rw-r--r-- 1 lakshmanan users 62 2014-10-07 20:04 hello.c
```

A *hello.c* fájl méreteit tanulmányozva látható, hogy a *-R* segítségével a fájl visszaállt az eredeti állapotába.

Feladatok

♠ **3.1.A és B feladat:** Írassa ki a számítógép processzorának vendor id-jét! Az elkészített *bash* héjprogram (*/home/laboruser/bin/3_1.sh*) csak a vendor id-t írja ki a standard kimenetre, semmi mást!

♠ **3.1.C és D feladat:** Írassa ki a számítógép processzorának frekvenciáját! Az elkészített *bash* héjprogram (*/home/laboruser/bin/3_1.sh*) csak a processzor névleges frekvenciáját (mértékegység nélkül) írja ki a standard kimenetre, semmi mást!

♠ **3.2. feladat:** Írjon egy *bash* scriptet (*/home/laboruser/bin/3_2.sh*), amely megszámlolja a bemenetként kapott fájlban az üres sorokat!

```
1 $ cat file_3_2 | ./3_2.sh
2 12
```

♠ **3.3.A és C feladat:** Írjon egy *bash* scriptet *3_3.sh* néven, amely paraméterként egy pozitív egész számot kapva kilistázza a standard bemenete kapott öt oszlopból álló adatfolyam azon sorait, amelyek a parancssori paraméterként megadott számmal osztható számmal kezdődnek, és teszi mindezt úgy, hogy közben felcseréli a második és az ötödik oszlopok tartalmát.

```
1 $ cat input.txt
2 1 a1 b c1 d
3 2 a2 b c2 d
4 3 a3 b c3 d
5 4 a4 b c4 d
6 5 a5 b c5 d
7 $ cat input.txt | ./3_3.sh 2
8 2 d b c2 a2
9 4 d b c4 a4
10 $ cat input.txt | ./3_3.sh 3
11 3 d b c3 a3
```

♠ **3.3.B és D feladat:** Írjon egy *bash* scriptet *3_3.sh* néven, amely paraméterként egy pozitív egész számot kapva kilistázza a standard bemenete kapott öt oszlopból álló adatfolyam azon sorait, amelyek a parancssori paraméterként megadott számmal osztható számmal kezdődnek, és teszi mindezt úgy, hogy közben összefűzi a negyedik és az ötödik oszlopok tartalmát.

```
1 $ cat input.txt
2 1 a1 b c1 d
3 2 a2 b c2 d
4 3 a3 b c3 d
5 4 a4 b c4 d
6 5 a5 b c5 d
7 $ cat input.txt | ./3_3.sh 2
8 2 a2 b c2d
9 4 a4 b c4d
10 $ cat input.txt | ./3_3.sh 3
11 3 a3 b c3d
```

♠ **3.4. feladat:** Írjon egy *bash* héjprogramot (*/home/laboruser/bin/3_4.sh*), amely futtatáskor soronként kiírja

1. a futtató felhasználó nevét,
2. az aktuális dátumot (éééé. hh. nn. formátumban),

3. a bejelentkezett felhasználókat (mindegyiket egyszer és külön sorba) és
4. a felhasználó által épp futtatott `3_4.sh` script pid-jét.

♠ **3.5. feladat (opcionális):** Írjon egy *bash* scriptet `3_5.sh` néven, amely a szabványos CSV formátumból konvertál a magyar Excel számára értelmezhető CSV formátumába, azaz az oszlopokat elválasztó vesszőket lecseréli pontosvesszőkre. Vigyázzon, hogy csak az oszlopokat elválasztó vesszőket módosítsa a program! A konvertálandó állományt a standard bemeneten kell fogadnia a programnak. A feladat megoldása során, az egyszerűség kedvéért, feltételezheti, hogy az oszlopok száma nem haladja meg a hármat, illetve, hogy az oszlopok csak az angol ábécé kis- és nagybetűit, valamint a `,;:?!()<>[]` karakterek tartalmazzák.

♠ **3.6. feladat:** Írjon egy *bash* scriptet `3_6.sh` néven, amely parancssori paraméterként megkapja két létező fájl nevét, és megállapítja, hogy hány különböző sor található bennük. A script futási eredményeként két sor jelenik meg a szabványos kimeneten: az első sor tartalmazza azoknak a soroknak a számát, amelyek benne vannak az elsőként megadott fájlban, de nincsenek benne a másodikban, a második sor pedig ugyanezt az adatot tartalmazza vica-versa.

diff

A 3.7-es feladatok teszteléséhez az alábbi formátumú, csoportok és értékpárok összerendelését tartalmazó fájl használható:

```
1 $ cat file_3_7
2 Item1,2,200
3 Item2,3,500
4 Item3,1,900
5 Item2,2,800
6 Item1,2,600
```

A fájl csak nullánál nem kisebb számokat tartalmaz!

♠ **3.7.A feladat:** Írjon egy *bash* scriptet `3_7.sh` néven, amely összegzi az egyes csoportokba tartozó számok szorzatának az összegét. A script futását az alábbi kódrészlet szemlélteti:

```
1 $ cat file_3_7 | ./3_7.sh
2 Item1: 1600
3 Item2: 3100
4 Item3: 900
```

♠ **3.7.B feladat:** Írjon egy *bash* scriptet `3_7.sh` néven, amely kiírja minden csoport esetén a harmadik oszlopban található legkisebb elemet. A kimenet formátuma kövesse az alábbi sémát:

```
1 $ cat file_3_7 | ./3_7.sh
2 Item1: 200
3 Item2: 500
4 Item3: 900
```

♠ **3.7.C feladat:** Írjon egy *bash* scriptet *3_7.sh* néven, amely kiírja minden csoport első előfordulását. Azaz a működése és kimeneti formátuma legyen:

```
1 $ cat file_3_7 | ./3_7.sh
2 Item1: 2, 200
3 Item2: 3, 500
4 Item3: 1, 900
```

♠ **3.7.D feladat:** Írjon egy *bash* scriptet *3_7.sh* néven, amely megszámolja majd kiírja az egyes csoportok előfordulásának a számát, azaz:

```
1 $ cat file_3_7 | ./3_7.sh
2 Item1: 2
3 Item2: 2
4 Item3: 1
```

♠ **3.8 feladat (opcionális):** Írjon egy *bash* scriptet *3_8.sh* néven, amely egy három oszlopból álló (az oszlopokat minden esetben egy pontosvessző választja el) állomány második oszlopában szereplő 0 és 1 közötti számoknak 0.1 széles intervallumokba osztja, majd kiírja, egymástól vesszővel elválasztva, az egyes intervallumok gyakoriságát.

4 — A Monit

A *Monit* egy folyamatok, programok, fájlok, könyvtárak és fájlrendszerek monitorozására és menedzselésére használható eszköz Unix és Linux rendszereken.

A *Monit* egy egyszerű konfigurációs állománnyal vezérelhető, amelynek szabad formátuma és token orientált szintaxisa van [13]. A *Monit* logokat készít, illetve személyre szabható üzenetekben e-mailen is riaszt. Mindezek mellé a *Monit* egy HTTP(S) interfésszel is rendelkezik, így böngészővel is lekérdezhető az állapota.

```

1 $ cat /var/log/syslog | more | fold -w 80 -s
2 Apr 15 06:38:42 irulabor monit[15900]: Shutting down Monit HTTP server
3 Apr 15 06:38:43 irulabor monit[15900]: Monit HTTP server stopped
4 Apr 15 06:38:43 irulabor monit[15900]: Monit daemon with pid [15900] stopped
5 Apr 15 06:38:43 irulabor monit[15900]: 'irulabor.irulabor' Monit 5.15 stopped
6 Apr 15 06:39:54 irulabor monit[17189]:
7     Starting Monit 5.15 daemon with http interface at [localhost]:2812
8 Apr 15 06:39:54 irulabor monit[17192]:
9     Starting Monit HTTP server at [localhost]:2812
10 Apr 15 06:39:54 irulabor monit[17192]: Monit HTTP server started
11 Apr 15 06:39:54 irulabor monit[17192]: 'irulabor.irulabor' Monit 5.15 started
12 Apr 15 06:39:54 irulabor monit[17192]:
13     'irulabor.irulabor' swap usage of 36.4% matches resource limit [swap usage<25.0%]
14 Apr 15 06:39:54 irulabor monit[17192]:
15     'irulabor.irulabor' mem usage of 77.0% matches resource limit [mem usage<25.0%]
16 Apr 15 06:40:24 irulabor monit[17192]:
17     'irulabor.irulabor' swap usage of 36.4% matches resource limit [swap usage<25.0%]
18 Apr 15 06:40:24 irulabor monit[17192]:
19     'irulabor.irulabor' mem usage of 77.0% matches resource limit [mem usage<25.0%]
20 Apr 15 06:40:24 irulabor monit[17192]:
21     'irulabor.irulabor' cpu usage of 44.1% matches resource limit [cpu usage<33.0%]
22 Apr 15 06:40:54 irulabor monit[17192]:
23     'irulabor.irulabor' swap usage of 36.4% matches resource limit [swap usage<25.0%]
24 Apr 15 06:40:54 irulabor monit[17192]:
25     'irulabor.irulabor' mem usage of 77.1% matches resource limit [mem usage<25.0%]
26 Apr 15 06:40:54 irulabor monit[17192]:
27     'irulabor.irulabor' cpu usage of 43.5% matches resource limit [cpu usage<33.0%]
28 Apr 15 06:40:55 irulabor monit[17192]: 'irulabor.irulabor' exec: /bin/bash
29 Apr 15 06:41:25 irulabor monit[17192]:
30     'irulabor.irulabor' swap usage of 36.4% matches resource limit [swap usage<25.0%]
31 Apr 15 06:41:25 irulabor monit[17192]:
32     'irulabor.irulabor' mem usage of 76.9% matches resource limit [mem usage<25.0%]
33 Apr 15 06:41:25 irulabor monit[17192]:
34     'irulabor.irulabor' cpu usage check succeeded [current cpu usage=1.3%]

```

A monit indítása

A *Monit* viselkedése parancssori paraméterekkel, illetve a konfigurációs állományával (*monitrc*) szabályozható. A parancssori paraméterek felülírják a konfiguráció állományban található beállításokat. Ha a konfigurációs állomány nem található az alapértelmezett elérési útvonalak [13] egyikén sem, akkor annak elérési útvonala megadható parancssori paraméterként is:

```
1 $ monit -c /var/monit/monitrc
```

A *Monit* elindítása előtt célszerű ellenőrizni, hogy a konfigurációs állomány tartalmaz-e szintaxisbeli hibákat:

```
1 $ monit -t
2 $ Control file syntax OK
```

Hiba esetén a *Monit* hibáüzenetet ír ki a képernyőre, amely tartalmazza a hibát tartalmazó sor számát is. Amint sikerült a hibákat kijavítani, a *Monit*ot el lehet indítani:

```
1 $ monit
```

A *Monit* le fogja magát választani a terminálról, és háttérfolyamatként, „démonként”, fog futni.

Az első lépések

Habár a *Monit* tartalmaz egy minta konfigurációs állományt, annak nagy része csak mintaként szolgál, ki van kommentelve. A használat előtt így szükség lehet a beállítások finomhangolására, a szükséges helyekről a # eltávolítása az elvárt működés érdekében. Talán a legfontosabb ilyen részek a mailszerver, illetve a riasztásokra használandó e-mail cím megadása.

set mailsERVER

A mailszerver adatait, amelyet a *Monit*nak a riasztások elküldéséhez kell használnia, a `set mailsERVER` paranccsal lehet megadni:

```
1 SET MAILSERVER
2     <hostname|ip-address>
3     [PORT number]
4     [USERNAME string] [PASSWORD string]
5     [using SSL [with options {...}]]
6     [CERTIFICATE CHECKSUM [MD5|SHA1] <hash>],
7     ...
8     [with TIMEOUT X SECONDS]
9     [using HOSTNAME hostname]
```

Több mailszerver is megadható, ebben az esetben vesszővel kell elválasztani őket. Ha a *Monit* nem tud csatlakozni az elsőhöz, akkor a listán a sorra következővel próbálkozik.

```
1 set mailsERVER smtp.gmail.com, smtp.other.host
```

set alert

A `set alert` parancs azt az e-mail címet adja meg, ahová riasztás esetén levelet kell küldeni:

```
1 set alert foo@bar
```

Több e-mail cím esetén mindet külön-külön definiálni kell a `set alert` parancs segítségével.

Előfordulhat, hogy riasztások szintjén szeretnénk definiálni, hogy az adott riasztást kinek küldje el a *Monit*. Ilyen, ún. lokális, riasztás esetében a címzett a riasztás definíciójánál adható meg:

```

1 check host myhost with address 1.2.3.4
2   if failed port 3306 protocol mysql then alert
3   if failed port 80 protocol http then alert
4   alert foo@baz #Local service alert

```

De globálisan is beállítható, hogy melyik címre milyen típusú riasztások legye-
nek elküldve:

```

1 set alert foo@bar only on { timeout, nonexist }
2 set alert foo@baz but not on { instance }

```

Konfigurációs mintapéldák

A következőkben a *Monit* használatára láthatunk pár mintapéldát [14].

Szolgáltatások elérhetőségének a tesztelése

Teszteljük le, hogy elérhető-e még a nemrég létrehozott *irulabor.vmware* oldal. Ehhez a konfigurációs állományhoz adjuk hozzá egy ellenőrzést, amely megpróbál „csatlakozni” az oldalhoz HTTP majd HTTPS kapcsolaton keresztül:

```

1 check host irulabor with address irulabor.vmware
2   if failed port 80 protocol http then alert
3   if failed port 443 protocol https then alert

```

Hasonló módon tudjuk ellenőrizni, hogy fut-e a MySQL szerver a rendszerünkön. A ping parancs segítségével ellenőrizhetjük azt is, hogy a munkaállomás elérhető-e:

```

1 check host localhost with address 127.0.0.1
2   if failed ping then alert
3   if failed port 3306 protocol mysql then alert

```

Hajnali 4 óra – szolgáltatások automatikus kezelése

Ha valami nem működik az elvártaknak megfelelően, akkor két megoldás van: előkapjuk a kalapácsot és törünk-zúzunk, vagy újraindítjuk a gépet. Meglepőnek tűnhet, de ez utóbbi megoldás gyakran beválik.

A *monit* automatikusan végre tudja hajtani nekünk a szolgáltatások újraindítását, ha azok nem működnek az elvárásoknak megfelelően. Így például a következő sorok az Apache szervert indítják újra, ha az valamiért nem fogadja a HTTP kéréseket:

```

1 check process apache with pidfile /var/run/httpd.pid
2   restart program = "/etc/init.d/apache2 restart"
3   if failed port 80 protocol http then restart

```


Mielőtt újraindítanánk egy szolgáltatást, gyakran meg szeretnénk bizonyosodni róla, hogy tényleg az tényleg problémába ütközött, és nem csak egy fals pozitív hibaiüzenetet látunk. A *monit*nak megmondhatjuk azt is, hogy csak akkor indítsa újra a szolgáltatás, ha az hosszabb ideig nem válaszol:

```
1 if failed port 80 protocol http for 2 cycles then restart
```

Programok ellenőrzése

A *Monit*tal lehetőség nyílik felhasználó program futtatására, majd a program eredményétől függően akciók végrehajtására. Ez a gyakorlatban azt jelenti, hogy a *monit* bármilyen ellenőrzést végre tud hajtani, csak a megfelelő héjprogramot kell elkészíteni hozzá.

Legyen a *sales.sh* egy bash script, amely csatlakozik a rendszeren futó MySQL adatbázishoz, majd különböző lekérdezéseket futtat le. Ha mindent rendben talál, akkor futási eredményként 0-t, ellenkező esetben 1-et ad vissza.

```
1 check program salesreport with path /var/monit/programs/sales.sh
2   every "* 8 * * 1-5"
3   if status != 0 then alert
```

Feladatok

4.1. feladat: Határozza meg a gépen futó leginkább processzor igényes alkalmazást!

`top`

♠ **4.2. feladat:** Töltse le a virtuális gépen található operációs rendszer verziójának megfelelő *monit* programot a `/root/bin` könyvtárba (A labor során használja a tantárgy honlapján megtalálható verziót!). A program elérési útvonala legyen `/root/bin/monit/bin`! Indítsa el, majd jelentkezzen be a program webes felületén a `admin/monit` alapértelmezett felhasználónév/jelszó kombinációval. Ezen adatokat a feladatok során ne változtassa meg! Ügyeljen arra, hogy a *monit* parancssori paraméterek megadása nélkül is indítható legyen!

<https://mmonit.com/monit/>

♠ **4.3. feladat:** Módosítsa a *monit* konfigurációs állományát, hogy jelezzen, ha a memóriahasználat túllépi a 25%-ot. A problémáról a program küldjön e-mailt a `laboruser@irulabor.irulabor` címre. Az e-mail tartalmazza az alapértelmezett mezőkön kívül az Ön NEPTUN kódját a

```
1 Neptun: NEPTUN
```

formátumban rögtön a hostot megadó sor után! Ellenőrizze, hogy az e-mail megérkezett!

`cat /var/mail/laboruser`

4.4. feladat: Töltse le a virtuális gép Linux disztribúciójának és verziójának megfelelő *stress* és *cpulimit* programokat. A letöltött programok segítségével írjon egy bash scriptet, amely 25%-ig leterheli a CPU-t.

```
stress -c 1 & cpulimit -p $( pidof -o $! stress ) -l 25
```

4.5. feladat: Módosítsa a *monit* konfigurációs állományát úgy, hogy a monitorozó program jelezzen, ha olyan futó alkalmazást észlel, amely legalább 33%-ig leterheli a CPU-t. A problémáról a *laboruser@irulabor.irulabor* e-mail címre küldött levél tartalmazza, az alapértelmezett mezőkön kívül, az Ön NEPTUN kódját a

```
1 Neptun: NEPTUN
```

formátumban rögtön a munkaállomást megadó sor után! Ellenőrizze, hogy az e-mail megérkezett!

♠ **4.6. feladat:** Módosítsa a *monit* konfigurációs állományát, hogy a monitorozó program termináljon minden olyan futó alkalmazást, amely legalább 33%-ig leterheli a CPU-t. Tesztelje le, hogy működik-e az Ön által írt feltétel!

Hivatkozások

- [1] Tony Bautts, Terry Dawson, Gregor N. Purdy, *Linux hálózati adminisztrátorok kézikönyve*, Kossuth Kiadó, 2005., ISBN: 9630947498
- [2] Andrew Ford, *Apache 2 Pocket Reference: For Apache Programmers & Administrators*, O'Reilly Media, 1 edition, 2008., ISBN-10: 059651889
- [3] Arnold Robbins, Nelson H.F. Beebe, *Classic Shell Scripting*, O'Reilly Media, 1st edition, 2005., ISBN-10: 0596005954
- [4] John Bambenek, Agnieszka Klus, *Grep Pocket Reference*, O'Reilly Media, 1 edition, 2009., ISBN-10: 0596153600
- [5] Arnold Robbin, *sed and awk Pocket Reference*, O'Reilly Media, 2nd Edition, 2002., ISBN-10: 0596003528
- [6] Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley, *Unix and Linux System Administration Handbook* Prentice Hall, 2010., ISBN-10: 0131480057
- [7] Ubuntu súgó,
<http://sugo.ubuntu.hu/>
- [8] Ubuntu kiszolgálók kézikönyve,
<http://sugo.ubuntu.hu/10.10/html/serverguide/hu/serverguide.pdf>
- [9] AddUsersHowto,
<https://help.ubuntu.com/community/AddUsersHowto>
- [10] IptablesHowTo,
<https://help.ubuntu.com/community/IptablesHowTo>
- [11] LDAP,
<http://unixlinux.tmit.bme.hu/LDAP>
- [12] Ubuntu httpd,
<https://help.ubuntu.com/10.04/serverguide/httpd.html>
- [13] Monit manual,
<https://mmonit.com/monit/documentation/monit.html>
- [14] Monit,
<https://mmonit.com/monit/>

5 — Appendix

Linux héjak és beépített parancsok

Néhány közkedvelt Linux héj:

| | |
|--------------|--|
| <i>bash</i> | a GNU <i>Bourne-Again Shell</i> je |
| <i>ksh</i> | a <i>Korn shell</i> , az eredeti vagy egy klónja |
| <i>pdksh</i> | a <i>Public Domain Korn</i> héj |
| <i>sh</i> | az eredeti <i>Bourne shell</i> |
| <i>zsh</i> | a <i>Z shell</i> |

Fontosabb beépített parancsok:

| | |
|--------------|--|
| <i>.</i> | beolvassa és végrehajtja az aktuális fájlt |
| <i>cd</i> | megváltoztatja az aktuális könyvtárat |
| <i>eval</i> | végrehajtja a szöveggént megadott héjprogramot |
| <i>exit</i> | kilép a héjprogramból |
| <i>read</i> | beolvas egy értéket az inputról |
| <i>test</i> | kiértékeli a paraméterként megadott kifejezést |
| <i>unset</i> | „törli” a héj egy változóját vagy függvényét |

A mindennapi életben hasznosnak bizonyult utasítások:

| | |
|-----------------|---|
| <i>basename</i> | kiírja az útvonal utolsó komponensét opcionálisan a suffix elhagyásával |
| <i>dirname</i> | kiírja az útvonalat az utolsó elemét leszámítva |
| <i>id</i> | kiírja a felhasználó azonosítóját és nevét |
| <i>date</i> | kiírja az aktuális dátumot és időt a paraméterekben specifikált formátumban |
| <i>who</i> | a bejelentkezett felhasználók kilistázása |
| <i>stty</i> | az aktuális terminál beállításainak a kezelése |

Szövegkezelés

| | |
|---------------|--|
| <i>awk</i> | szöveges állományok feldolgozására reguláris kifejezések segítségével |
| <i>cat</i> | fájlkonkatenáció |
| <i>cmp</i> | egyszerű program fájlok összehasonlítására |
| <i>dd</i> | blokk szintű adatmozgatás |
| <i>echo</i> | argumentum kiírása a standard kimenetre |
| <i>egrep</i> | kibővített <i>grep</i> , amely az ERE típusú reguláris kifejezéseket használja |
| <i>expand</i> | tabulátorok szóköz karakterekre cseréje |
| <i>fgrep</i> | gyors <i>grep</i> |
| <i>grep</i> | <i>g/re/p</i> :) |
| <i>less</i> | hosszú fájlok lapozása („Less is more.”) |
| <i>more</i> | az eredeti BSD Unix lapozó program |
| <i>sed</i> | karakterfolyamok módosítása |
| <i>sort</i> | szöveges fájlok rendezése |
| <i>spell</i> | helyesírás-ellenőrző |
| <i>tee</i> | a sztenderd bemenetet a sztenderd kimenetre és a megadott fájlba másolja |
| <i>uniq</i> | duplikált sorokat eltávolítása rendezett bemenetből |
| <i>wc</i> | sorok, szavak és a karakterek megszámlálása a bemenetben |

Fájlkezelés

| | |
|---------------------|--|
| <i>chgrp</i> | fájlok és könyvtárak csoportjának a megváltoztatása |
| <i>chmod</i> | fájlok és könyvtárak hozzáférési jogának a megváltoztatása |
| <i>chown</i> | fájlok és könyvtárak tulajdonosának a megváltoztatása |
| <i>cp</i> | fájlok és könyvtárak másolása |
| <i>df</i> | üres helyek mérete a háttértárolón |
| <i>diff</i> | fájlok összehasonlítása |
| <i>du</i> | a diszkek foglaltsági adatainak a megjelenítése |
| <i>gzip, gunzip</i> | tömörítő program és kitömörítő programok |
| <i>head</i> | fájlok első <i>n</i> sorának a listázása |
| <i>locate</i> | fájl keresése a neve alapján |
| <i>ls</i> | fájlok listázása |
| <i>md5sum</i> | ellenőrző összeg számolása MD5 algoritmussal |
| <i>mkdir</i> | új könyvtár készítése |
| <i>pwd</i> | aktuális könyvtár kiírása |
| <i>rm</i> | fájlok és könyvtárak törlése |
| <i>rmdir</i> | üres könyvtárak törlése |
| <i>sha1sum</i> | ellenőrző összeg számolása SHA1 algoritmussal |
| <i>tail</i> | fájlok utolsó <i>n</i> sorának a kiírása |
| <i>tar</i> | szalagarchiváló |
| <i>touch</i> | fájlok hozzáférési idejének a módosítása |
| <i>umask</i> | alapértelmezett hozzáférés beállítása fájlok számára |

Folyamatkezelés

| | |
|--------------|---|
| <i>fuser</i> | adott fájlt vagy szocketet használó folyamatok megkeresése |
| <i>kill</i> | jelzés küldés egy vagy több folyamatnak (tipikusan „kill” küldése, hogy fejezze be a futását) |
| <i>nice</i> | folyamatok prioritásának a megváltoztatása az elindításuk előtt |
| <i>ps</i> | információ a futó folyamatokról |
| <i>sleep</i> | a végrehajtás felfüggesztése a megadott időre |
| <i>top</i> | a leginkább CPU-igényes folyamatok listázása |

Egyébb programok

| | |
|---------------|---|
| <i>man</i> | utasítás, függvény, rendszerhívás, stb. manuáljának a listázása, a manuálból a q betű-vel lehet kilépni |
| <i>scp</i> | biztonságos távoli fájlmásolás |
| <i>ssh</i> | „secure shell” |
| <i>uptime</i> | megadja a legutóbbi bekapcsolás óta eltelt időt, illetve a rendszer terheltségi adatait |