Információs rendszerek üzemeltetése
– Linux admin –

BME TMIT

2019

Oldalszám

0 — Bevezetés

Ez a labormérés három 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 néggyel 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. Ehhez a `https://github.com/ng201/iru` címről először le kell tölteni az `iru-data.iru` állományt, és azt a virtuális gépen a `/root` könyvtárba kell másolni. Ezután rendszergazdaként bárhol kiadva a `iru-test NEPTUN` parancsot az ellenőrző script elindul. Lehetőség van egyes feladatcsoportok külön ellenőrzésére is, ekkor a NEPTUN kód előtt a feladatcsoport sorszámát is meg kell adni, tehát például a második feladatcsoport ellenőrzéséhez a `iru-test 2 NEPTUN` parancs kiadása 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:

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

- 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 „almafa”, akkor az használható „AlMafA”-ként is, de akkor végig csak ebben a formában).

- Amennyiben futtatható héjprogramot, szkriptet kell írni, akkor annak nevét származtassa a feladat sorszámából az alábbi módon:
  `3.4.A feladat` → `3_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` home könyvtárában található `bin` könyvtárába. Amennyiben ez a könyvtár nem létezik, akkor hozza létre. A scripteknek ne feledjen futtatási jogokat adni!

- 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 rendszeradminisztrátorok 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 átemelhetőek 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, mi a futtatott Debian rendszer kódneve!
lsb_release -da

1.2. feladat: Írassa ki a telepített csomagokat! Telepítve van a Midnight Commander fájlkezelő a rendszerben?
dpkg -l

1.3. feladat: Nézze meg, hogy milyen hálózati interfészei vannak a virtuális gépnek. Milyen IP címekre vannak konfigurálva az egyes interfészek?
ip addr

♠ **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!
sudo

♠ **1.8. feladat:** A biztonság érdekében tiltsa le a *root* felhasználó SSH-n történő bejelentkezésének jogát!
mc, */etc/ssh/sshd_config*, *systemctl restart ssh*

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! Nézze meg, hogy a szerver melyik verzióját sikerült installálnia! Állítsa be, hogy a rendszergazda csak a „root” jelszóval léphessen be. Ne feledje letiltani a socket alapú azonosítást sem!
mysql_secure_installation, *UPDATE user ...*

1.11. feladat: Derítse ki az előző feladatban telepített adatbázis-szerver nevét, illetve vizsgálja meg a beállításait! Derítse ki, hogy milyen porton figyel a telepített adatbázis-szerver.
mysql -v

1.12. feladat: Az *lsof* 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 virtuális gépen kívülről, tehát a helyi hálózatból, vagy bármilyen más IP címről!
man lsof, *lsof -i -P*

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 töltse fel a MySQL szerverre.

1.15. feladat: SQL parancsok segítségével ismerje meg az előző feladatban feltöltött egyszerű adatbázist! 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 fájlok és könyvtárak között vannak [1, 2]:

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.

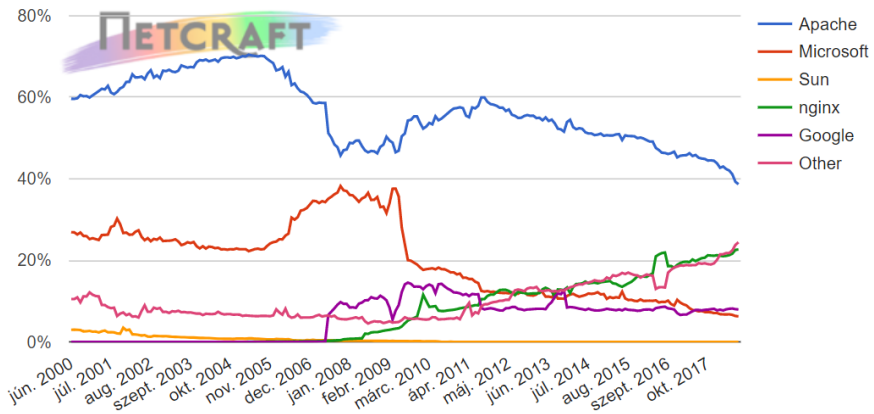
sites-enabled — A `mods-enabled` mintájára a `sites-enabled` szimbolikus linkeket tartalmaz az `/etc/apache2/sites-available` könyvtárra.

Az Apach 2 induláskor beolvassa a MIME-dokumentumtípusokat tartalmazó fájlt is[1], ennek nevét a `TypesConfig` direktíva adja meg. Ennek alapértelmezett értéke a `/etc/mime.types`.

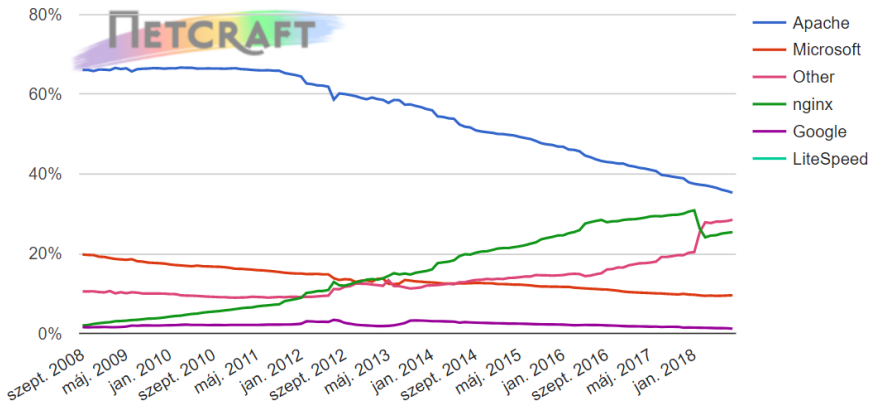
Apache modulok

Az Apache 2 egy moduláris kiszolgáló. Ez azt jelenti, hogy a kiszolgáló magja 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 engedélyezhetőek, ellenkező esetben az Apache 2-t újra kell fordítani a modulok hozzáadásához vagy eltávolításához [2].

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



2. ábra. A (aktív) HTTP szerverek piaci részesedése a Netcraft mérései alapján [3]



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

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

```
# apt-get install libapache2-mod-auth-mysql
```

Modulok engedélyezése az `a2enmod` segédprogram segítségével lehetséges:

```
# a2enmod auth_mysql
# /etc/init.d/apache2 restart
```

Hasonlóképpen az `a2dismod` segítségével letilthatók az egyes modulok:

```
# a2dismod auth_mysql
# /etc/init.d/apache2 restart
```

Virtuális kiszolgálók

Az Apache 2 egyik leghatékonyabb szolgáltatása, hogy egyetlen gépen több webkiszolgálót is képes futtatni [2]. Ehhez a `VirtualHost` szolgáltatást lehet igénybe venni. Alapértelemben az Apache 2-ben egyetlen virtuális kiszolgáló van beállítva, ennek adatait a `/etc/apache2/sites-available/000-default.conf` fájl tartalmazza.

Új virtuális kiszolgáló létrehozásához ezt a fájlt lemásolni és célszerű mintaként használni. A fájlban található beállítások csak az adott virtuális kiszolgálóra lesznek érvényesek. Ha egy direktíva nincs megadva a virtuális kiszolgálóra, akkor a globális, rendszerszintű (alapértelmezett) beállítás kerül felhasználásra.

A `DocumentRoot` direktíva megadja, hogy az Apache 2 hol keresse a webhelyet felépítő fájlokat.

A `ServerName` (elhagyható) direktíva megadja, hogy a webhely mely FQDN-re válaszoljon. Az alapértelmezett virtuális kiszolgálóhoz nincs megadva ez a 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.

```
ServerName irulabor.vmware
ServerAlias *.irulabor.vmware
```

A virtuális kiszolgáló beállító állománya tartalmazhat `Directory`, `Files` és a `Location` blokkokat is, amelyek egy-egy könyvtárra (beleértve az abban levő alkönyvtárakat és fájlokat), fájlra, illetve URL-re szűkítve adnak meg konfigurációkat:

```
<Directory "/home/laboruser/public_html">
   AllowOverride None
   Require all granted
</Directory>
```

Reguláris kifejezések is használhatóak a útvonal leírásához, ekkor azonban a ~ előtag használata kötelező:

```
<Files "cat.html">
   # a cat.html fájlra érvényes beállítások
   Require all granted
</Files>

<Files ~ "\.(gif|jpe?g|png)$">
   Require all denied
</Files>
```

A <Location> direktíva olyan esetekben használatos, amikor a tartalom a fájlrendszeren kívül található. A fájlrendszerben található tartalmakhoz való hozzáférés szabályozására a <Directory> és a <Files> direktívák ajánlottak. Ez alól kivételt képezhet a <Location "/"> direktíva, ami a teljes webszerverre érvényes konfigurációkat tartalmazza.

Az /etc/apache2/sites-available könyvtár tartalmát 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:

```
# a2ensite irulabor
# systemctl restart apache2
```

Hasonlóképpen az a2dissite segédprogrammal tiltható le egy webhely:

```
# a2dissite irulabor
# systemctl restart apache2
```

Az Apache biztonsági beállításai

Hozzáférések szabályozása

Egy oldalhoz vagy annak egy részéhez a látogatók IP címe alapján történő hozzáférés szabályozása a mod_authz_host modul használható. A szabályok, amelyek IP címek, illetve hosztnevek alapján tesznek lehetővé szűrést, Require direktívák segítségével adhatóak meg:

```
Require host address
Require ip ip.address
```

Ilyen direktívákat el lehet helyezni mind a <Directory>, mind a <Files>, mind pedig a <Loaction> szakaszokban is.

A legegyszerűbb beállítások alább láthatóak:

- minden kérés tiltása

```
Require all denied
```

- minden kérés engedélyezése

```
Require all granted
```

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

```
Require host example.org
```

A *RequireAll*, *RequireAny*és a *RequireNone* direktívák segítségével bonyolultabb feltételek készíthetők több *Require* direktíva egy egységbe foglalásával:

```
<RequireAll>
   Require all granted
   Require not ip 10.252.46.165
</RequireAll>
```

Apache autentikáció

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

```
AuthType Basic
AuthName "IRULabor - vedett"
AuthBasicProvider file
AuthUserFile "/etc/apache2/passwd/.passwords"
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 ellesett példát:

```
<Directory /home/laboruser/public_html/iru/vedett>
   AllowOverride All
      AuthType Basic
      AuthName "IRULabor - vedett"
      AuthBasicProvider file
      AuthUserFile "/etc/apache2/passwd/.passwords"
      Require valid-user
</Directory>
```

A *htpasswd* fájlt (*/etc/apache2/passwd/passwords* 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:

```
# htpasswd /etc/apache2/passwd/.passwords ujfelhasznalo
New password:
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
```

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 felülbírálását. 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-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 gép a* `irulabor.vmware` *nevet a* 127.0.0.1 *címre oldja fel.*

♠ **2.4. feladat:** A fizikai gépen (és nem a virtuális gépen futó linuxon) töltse 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, a 127.0.0.1 címről 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. 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* fájllal. Mire alkalmas ez a fájl?

♠ **2.8. feladat:** Az alapértelmezett webkiszolgáló konfigurációban ennek a fájlnak 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 [4].

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:

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

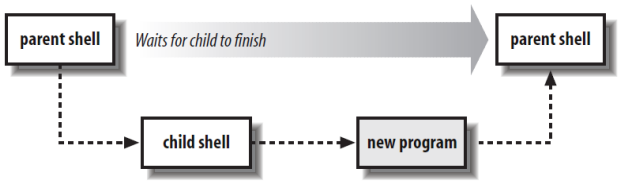
Egy,kettő, három, négy, öt... Egy nagy, többfelhasználós rendszerben azonban a lista hosszúra nyúlhat, így a sorok megszámlálása nehézkessé válhat. Itt a lehetőséget a feladat automatizálására.

A bejelentkezett felhasználók megszámolásához módosítsunk egy picit a programon, használjuk fel a *wc* programot, amely megszámolja a bemenetén a sorokat, szavakat és karaktereket. A két program „összekapcsolására” használjuk a *|*-t:

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

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

```
$ cat > nusers
who | wc -l
^D
$ chmod +x nusers
$ ./nusers
5
```



4. ábra. Héjprogramok futtatása [4]

#!

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, karakterfüzérek 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 kertes ház."
4 volt egyszer egy szép kis kertes 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 rábírni a szöktetett karakterek értelmezésére:

```
$ echo "\n\n\n\n"
\n\n\n\n
$echo -e "Frakk\nSzerenke\nLukrecia"
Frakk
Szerenke
Lukrecia
```

A *cut*

A `cut` segítségével előre megadott karakterek mentén oszlopokra bonthatjuk a bemeneti állományt.

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

A *date* parancs

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

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

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

```
$ date +"%b"
Jan
$ date +"%B"
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.

```
$ envaltozom=Frakk_a_macskak_reme
$ echo $envaltozom
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:

```
kutya=Frakk macska1=Szerenke macska2=Lulkrecia
# ^^ ertekadas tobb valtozonak egy sorban
bacsi="Karoly bacsi"
# ^^ ureskarakterek eseten aposztrofot kell hasznalni
```

Változók értékeinek a konkatenációja is gyerekjáték:

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

További példákat változók értékadására az alábbi héjprogramban láthatunk:

```
#!/bin/bash

a=375
hello=$a

echo $hello         # 375
# hivatkozás a változóra

echo ${hello}       # 375
# mint elöbb

# idézőjelek...
echo "$hello"       # 375
echo "${hello}"     # 375
echo '$hello'       # $hello
# a valtozó behelyettesítéset letiltotta az aposztróf

u=`who | wc -l`
# ^^^ a parancs futási eredménye a változóba kerül

u=$( who | wc -l )
# ^^^ 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, túlcsordulás ellenőrzése nélkül. Az operátorok és a rájuk érvényes szabályok a C nyelv szabályait követik.

```
$ x=100
$ y=3
$ echo $((x/y))
33
$ echo $((y/x))
0
$ echo $((++x))
101
```

Parancsok futtatási eredményének tárolása változóban

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

```
x=`echo "12"`
y=$(echo "almafa")
echo x # 12
echo y # almafa
OUTPUT="$(ls -1)" # több sor van a kimenetben
echo "${OUTPUT}"
```

Elágazások

Az elágazások *bash* formája a következő:

```
if [expression];
then

kód, ha a kifejezés igaz

fi
```

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

```
#!/bin/bash

inp1=12
inp2=11
echo "1. Összeadás"
echo "2. Kivonás"
echo "3. Szorzás"
echo -n "Válasszon egy aritmetikai műveletet [1,2 vagy 3]? "
read oper

if [ $oper -eq 1 ]
then
      echo "Az összeadás eredménye: " $(($inp1 + $inp2))
else
      if [ $oper -eq 2 ]
      then
            echo "A kivonás eredménye: " $(($inp1 - $inp2))
      else
            if [ $oper -eq 3 ]
            then
                  echo "A szorzás eredménye: " $(($inp1 * $inp2))
            else
                  echo "Nem beszélni magyar?"
            fi
      fi
fi
```

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

```
#!/bin/bash
#!/bin/bash

echo "1. szám: "
read a
echo "2. szám: "
read b

if [ "$a" -eq "$b" ];
then
   echo "a két szám egyforma";
else
   if [ "$a" -lt "$b" ];
   then
      echo "az elsȍ szám a kisebb"
   else
      echo "az elsȍ szám a nagyobb "
   fi
fi

## -eq # equal
## -ne # not equal
## -lt # less than
## -le # less than or equal
## -gt # greater than
## -ge # greater than or equal
```

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

```
#!/bin/bash

month=$(date +"%b")
# echo "$month"

if [ $month == "Jan" ] || [ $month == "Mar" ] ||
   [ $month == "May" ] || [ $month == "Jul" ] ||
   [ $month == "Aug" ] || [ $month == "Oct" ] ||
   [ $month == "Dec" ]
then
   echo "31"
else
   if [ $month != "Feb" ]
   then
      echo "30"
   else
      echo "?????"
   fi
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 néggyel 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:

```
year=$( date +%Y )
sz=$((year%4))

if [ $sz -ne 0 ]
then
   echo "28"
else
   sz=$((year%400))

   if [ $sz -eq 0 ]
   then
      echo "29"
   else
      sz=$((year%100))
      if [ $sz -eq 0 ]
      then
         echo "28"
      else
         echo "29"
      fi
   fi
fi
```

Ciklusok

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

```
for arg in [list]
do
 utasítás(ok)...
done
```

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

```
#!/bin/bash
# Bolygok listazasa.
for planet in Merkur Venusz Fold Mars Jupiter Szaturnusz
          Uranusz Neptunusz Pluto
do
  echo $planet # soronkent egy-egy bolygó
done
echo; echo "Fene! A Pluto mar nem bolygo!"
```

```
9 exit 0
```

Parancssori argumentumok

A scripteket tetszőleges számú parancssori argumentummal meghívhatjuk. Ilyenkor a $1,$1,...,$9,${10},... 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 előzni 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 elölről 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 karakterfüzérek, 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 folyam megváltoztatását,

Karakter	BRE / ERE	Jelentése
`\`	mindkettő	általában a karakter speciális jelentését kapcsolja ki („szökteti”)
`.`	mindkettő	pontosan egy (de az lehet bármilyen) karakterre illeszkedik
`*`	mindkettő	nulla vagy több olyan karakterre illeszkedik, ami az őt megelőző mintával leírható (például a `.*` 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 \1 – \9 kifejezésekkel lehet hivatkozni
`\n`	BRE	visszaadja az n-edik ($n = 1, \ldots, 9$) zárójelbe zárt mintára illeszkedő karakterfüzért
`{n,m}`	ERE	éppen úgy viselkedik, mint az BRE `\{n,m\}`
`+`	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 [4]

- 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 fájlszerkesztők, mint például az *ed*, a *vi*, az *emacs*, a *vim*, stb.

A reguláris kifejezéseknek két típusa van használatban [4], a *POSIX BRE* (Basic Regular Expression)és a *ERE* (Extended Regular Expression), amelyek a soksok 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.

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 pontosa megegyeznek a „tolsztoj” füzérrel,

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

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

2. táblázat. Linux programok és az általuk használt reguláris kifejezések típusa [4]

`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, tolsztoj, tolKIAFENEtoj, stb.).

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 karakterfüzérek (sztringek) keresését reguláris kifejezések alapján [5].

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 folyományaként 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 [6]. Működésének lényege, hogy a feldolgozandó szöveget soronként egy átmeneti tárba, 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:

`p`	kiíratás,	`d`	törlés,
`s`	helyettesítés,	`a`	hozzáfűzés,
`i`	beszúrás,	`c`	a mintatér cseréje,
`y`	karakterek cseréje.		

A *sed*et leggyakrabban egy másik programtól érkező kimenet feldolgozására használjuk:

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

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

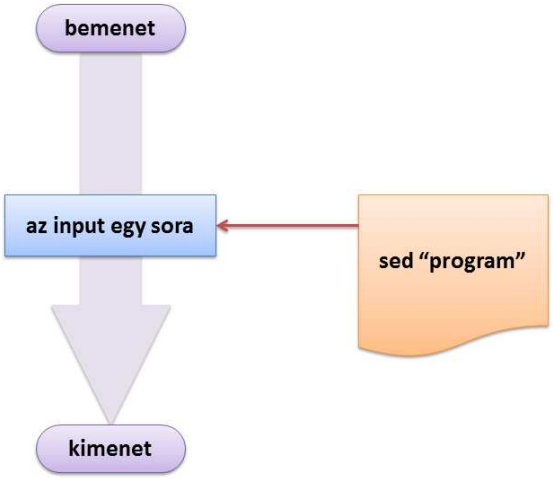
```
<cím1>,<cím2> parancs
```

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
   ```



5. ábra. A *sed* működési elve

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/'`
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 oszlopcsere a'la *sed*:

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

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

```
$ 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 [6, 4].

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 fájlnév 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 osztja. Alapesetben a rekordok sorok, a mezők pedig egy vagy több nem üres karakterből álló karakterfüzérek (stringek). Habár ezeket a beállításokat az *awk* programozó bármikor tetszés szerint megváltoztathatja [4].

Egy tipikus *awk* program a végrehajtása során a bemeneti adatokat egy másféle kimenetté 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 megszámolja, 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:

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

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

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

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

```
$ 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:

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

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

```
$ awk '
> BEGIN { print "Analysis of \"foo\"" }
> /foo/ { ++n }
> 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:

```c
#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:

```
$ diff -u hello.c hello_new.c > hello.patch
The above command will create a patch file named ``hello.patch''.

--- hello.c 2014-10-07 18:17:49.000000000 +0530
+++ hello_new.c 2014-10-07 18:17:54.000000000 +0530
@@ -1,5 +1,6 @@
 #include <stdio.h>

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

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:

```
patch -p[num] < patchfile
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 megegyezni. Figyeljük meg, hogy a patch fájl tartalmazza annak a fájlnak 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
```

```
patching file hello.c

$ ls -l hello.c
-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 megszámolja a bemenetként kapott fájlban az üres sorokat!

```
$ cat file_3_2 | ./3_2.sh
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. Ügyeljen arra, hogy nem minden sor első oszlopa tartalmaz egész számot, az ilyen sorokat nem szabad kiírnia!

```
$ cat input.txt
11 a1 b c1 d
08 a0 b c0 d
12 a2 b c2 d
13 a3 b c3 d
1a a0 b c0 d
14 a4 b c4 d
15 a5 b c5 d
$ cat input.txt | ./3_3.sh 2
12 d b c2 a2
14 d b c4 a4
$ cat input.txt | ./3_3.sh 5
15 d b c5 a5
```

♠ **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. Ügyeljen arra, hogy nem minden sor első oszlopa tartalmaz egész számot, az ilyen sorokat nem szabad kiírnia!

```
$ cat input.txt
11 a1 b c1 d
12 a2 b c2 d
1a a0 b c0 d
13 a3 b c3 d
14 a4 b c4 d
03 a0 b c0 d
15 a5 b c5 d
$ cat input.txt | ./3_3.sh 2
12 a2 b c2d
14 a4 b c4d
$ cat input.txt | ./3_3.sh 5
15 a5 b c5d
```

♠ **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),
4. a rendszer legutóbbi bekapcsolásának a napját és percre pontosan az időpontját (**éééé-hh-nn ÓÓ:pp** formátumban), valamint
5. 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:

```
$ cat file_3_7
Item1,2,200
Item2,3,500
Item3,1,900
Item2,2,800
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:

```
$ cat file_3_7 | ./3_7.sh
Item1: 1600
Item2: 3100
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:

```
$ cat file_3_7 | ./3_7.sh
Item1: 200
Item2: 500
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:

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

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

Hivatkozások

[1] Andrew Ford, *Apache 2 Pocket Reference: For Apache Programmers & Administrators*, O'Reilly Media, 1 edition, 2008., ISBN-10: 059651889

[2] Tony Bautts, Terry Dawson, Gregor N. Purdy, *Linux hálózati adminisztrátorok kézikönyve*, Kossuth Kiadó, 2005., ISBN: 9630947498

[3] Netcraft, *https://news.netcraft.com/archives/2018/08/24/august-2018-web-server-survey.html*

[4] Arnold Robbins, Nelson H.F. Beebe, *Classic Shell Scripting*, O'Reilly Media, 1st edition, 2005., ISBN-10: 0596005954

[5] John Bambenek, Agnieszka Klus, *Grep Pocket Reference*, O'Reilly Media, 1 edition, 2009., ISBN-10: 0596153600

[6] Arnold Robbin, *sed and awk* Pocket Reference, O'Reilly Media, 2nd Edition, 2002., ISBN-10: 0596003528

[7] Evi Nemeth, Garth Snyder, Trent R. Hein, Ben Whaley, *Unix and Linux System Administration Handbook* Prentice Hall, 2010., ISBN-10: 0131480057

[8] Iptables Essentials: Common Firewall Rules and Commands, *https://www.digitalocean.com/community/tutorials/iptables-essentials-common-firewall-rules-and-commands*

[9] Apache Basics: Installation and Configuration Troubleshooting, *https://www.digitalocean.com/community/tutorials/apache-basics-installation-and-configuration-troubleshooting*

4 — Appendix

Linux héjak és beépített parancsok

Néhány közkedvelt Linux héj:

`bash`	a GNU *Bourne-Again Shell*je
`ksh`	a *Korn shell*, az eredeti vagy egy klónja
`pdksh`	a *Public Domain Korn* héj
`sh`	az eredeti *Bourne shell*
`zsh`	a *Z shell*

Fontosabb beépített parancsok:

`.`	beolvassa és végrehajtja az aktuális fájlt
`cd`	megváltoztatja az aktuális könyvtárat
`eval`	végrehajtja a szövegként megadott héjprogramot
`exit`	kilép a héjprogramból
`read`	beolvas egy értéket az inputról
`test`	kiértékeli a paraméterként megadott kifejezést
`unset`	„törli” a héj egy változóját vagy függvényét

A mindennapi életben hasznosnak bizonyult utasítások:

`basename`	kiírja az útvonal utolsó komponensét opcionálisan a szuffix elhagyásával
`dirname`	kiírja az útvonalat az utolsó elemét leszámítva
`id`	kiírja a felhasználó azonosítóját és nevét
`date`	kiírja az aktuális dátumot és időt a paraméterekben specifikált formátumban
`who`	a bejelentkezett felahsználók kilistázása
`stty`	az aktuális terminál beállításainak a kezelése

Szövegkezelés

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

Fájlkezelés

`chgrp`	fájlok és könyvtárak csoportjának a megváltoztatása
`chmod`	fájlok és könyvtárak hozzáférési jogának a megváltoztatása
`chown`	fájlok és könyvtárak tulajdonosának a megváltoztatása
`cp`	fájlok és könyvtárak másolása
`df`	üres helyek mérete a háttértárolón
`diff`	fájlok összehasonlítása
`du`	a diszkek foglaltsági adatainak a megjelenítése
`gzip, gunzip`	tömörítő program és kitömörítő programok
`head`	fájlok első n sorának a listázása
`locate`	fájl keresése a neve alapján
`ls`	fájlok listázása
`md5sum`	ellenőrző összeg számolása MD5 algoritmussal
`mkdir`	új könyvtár készítése
`pwd`	aktuális könyvtár kiíratása
`rm`	fájlok és könyvtárak törlése
`rmdir`	üres könyvtárak törlése
`sha15sum`	ellenőrző összeg számolása SHA1 algoritmussal
`tail`	fájlok utolsó n sorának a kiíratása
`tar`	szalagarchiváló
`touch`	fájlok hozzáférési idejének a módosítása
`umask`	alapértelmezett hozzáférés beállítása fájlok számára

Folyamatkezelés

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

Egyébb programok

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