Óra1 Unix shellek és shell scriptek.

- a shell szerepe a unix környezetben
- standard unix shellek
- a shell script tulajdonságai
- shell script részei
- 1szerű shell script írás

Kinek való

- dolgozott már UNIX rendszeren, előbb utóbb találkozik vele
- ismerik a rendszer alapfilozófiáját
- nem feltétlen grafikus felületet akarnak használni a problémák megoldásához.
- Hatékonyan és gazdaságosan szeretnének üzemeltetni és menedzselni unix/linux gépeket

Szerepe

- Mi is az a shell: unix terminológia - az oprendszernek kiadott parancsokat beolvasó és értelmező programok
- Ez alapvető jellegzetessége minden unix rendszernek. Oka a rendszer felépítése, a rendszermag és köréje csoportosuló alkalmazások közötti kommunikáció

Shell= kagyló, burok, héj

- Elrejt vmit (a felhasználó elöl)
- A burok mindig valamely célközönség számára készül
- A fülhöz emelve a tenger zúgását halljuk (kommunikáció a felhasználóval)
- Van díszesebb és kevésbé díszes

Shellek

- Szabványos :sh, bash, ksh, csh, tcsh
- easy shell, pdmenu
- limited shell (dos szerű)
- router cli (cisco shell)
- screen
- stb , bárki írhat

A jövő (a shell kihalása)

- Számítási teljesítmény növekedése => grafikus felhasználói felületek elterjedése.
- Hátrány: felesleges erőforráshasználat, lassú kezelhetőség
- A shellekel (bár sok gyakorlati tudással) de gyorsabban érhetjük el céljainkat, +kényelmi funkciói is megjelentek

Szabványos unix shells

- Bourne (/bin/sh) →Bourne again shell

  Legősibb, alapvető kellék, mentes minden kényelmi funkciótól => nem kellenek csicsás dolgok ha csak programokat akarunk elindítani. (pl daemonok, scriptek, stb)
- Korn (/bin/ksh) → zsh

  A sh továbbfejlesztése. Sok már hiányzó kényelmi funkcióval.

  1szerűbb változó megadás (1 lépésben)

  be és kikapcsolható opciók

  helyettesítő nevek (aliasok) megjelenése

  parancsnaplózó (history)

  speciális válltozók
- C shell /bin/csh
- V shell /bin/vsh
- Midnight commander mc

A shell tulajdonságai

- Parancsértelmező és programozási nyelv. A parancsokat a standard inputról, vagy állományból olvassa.
- Egyszerűen kialakítható az ún. shell script vagy parancs file, amivel a parancskészlet bővíthető.
- A shell scriptek paraméterezhetősége teljesen megegyezik a programok paraméterezhetőségével.

A shell tulajdonságai

- Programnyelv, amely string változókra és hatékony vezérlési szerkezetekre épül.
- Egyzserü szintaxis a standard input-output átirányítására. (> <)
- Csővezeték segítségével komplex feladatok megoldása a meglévő segédprogramokkal. (|)

A shell tulajdonságai

- Sok DOS-ban megszerzett ismeret jól hasznosítható, de vannak eltérések. → a dos nem többfelhasználós. ??? Miért nem másolták jól le a Unix shellt a dos készítői.
- Beépített parancsok (cd, umask, echo,stb)
- Külső parancsok (kezdetben szinte minden külső parancs volt, kivéve ha nem lehetett máshogy megoldani „cd”)

A shell tulajdonságai

- Lehetővé teszi a processzek áttekinthető kezelését.
- A rendszer része, de egy új shell elkészítése nem igényel semmilyen rendszerprivilégiumot, bármikor lecserélhető.
- Egyszerűen konfigurálható a felhasználó igényei szerint:

A shell beállítása

- Speciális változók segítségével:
- $HOME: login katalógus neve
- $PATH: keresési út a programok végrehajtásához
- $CDPATH: keresési út a cd parancshoz
- $PS1: elsődleges prompt
- $PS2: másodlagos prompt
- $editor: alapértelmezett szövegszerkesztő
- Stb..

A shell beállítása

- Több jól definiált ponton speciális parancsállományok futtatására van lehetőség:
- login
- logout
- start

Shell-ek beálítása

Shell	Login	Start	Logout	rsh
sh	/etc/profile ~/.profile			
ksh	/etc/profile ~/.profile			
csh	/etc/csh.cshrc /etc/cshlogin ~/.cshrc -/.login	/etc/csh.cshrc ~/.cshrc	/etc/csh.logout ~/.logout	/etc/csh.cshrc ~/.cshrc
tcsh	/etc/csh.cshrc /etc/cshlogin ~/.tcshrc || ~/.cshrc ~/.login	/etc/csh.cshrc ~/.tcshrc ~/.cshrc	/etc/csh.logout ~/.logout	/etc/csh.cshrc ~/.cshrc
bash	/etc/profile ~/.bash_profile ~/.bash login ~/.profilé	~/.bashrc	~/.bash logout	

Kényelmi funkciók

- aliasok (alias ..=„cd ..”)
- history
- állománynév kiegészítés (tab)
- terminálkezelés (jogosultságok, átirányíthatóság, felfüggeszthetőség, msg…)
- Egymásba ágyazhatóság
- Online manual (man, apropos)
-

Parancsok

parancs kapcsolók argumentumok ....

- kapcsolók: parancs működését, eredményét befolyásoló paraméterek. Többnyire „-”szal kezdődnek (ls –al, ps –a)
- Argumentunok: filenév, vagy másik program bemenete
- Egy parancs több parancsot is tartalmazhat:
  - prog &
  - prog1; prog2
  - prog1&& prog2
  - prog1|| prog2
- Egy parancs több sorból is állhat. (folytató sor „\”)
- Lehetővé teszi, hogy az egyik program bemenete a másik kimenete legyen. (csővezeték, átirányítás)

File kezeléssel kapcsolatos parancsok

cd (change dir)	katalógus váltás
pwd (print w. dir.)	munkakatalógus kiírása
mkdir (make dir.)	katalógus létrehozása
rmdir (remove dir.)	katalógus törlése
ls (list)	katalógus lista
cp (copy)	fájl(ok) másolása
rm (remove)	fájl(ok) törlése
mv (move)	fájl(ok) áthelyezése
ln (link)	fájl hivatkozás készítés
ln-s	szimbolikus hivatkozás készítése

Renszer állapotával kapcs. parancsok

who	Aktuális felhasználók
ps	Futó processzek
kill	Szignál küldése
jobs	Háttérben levő feladatok listája
date	Dátum
top	Leg processzek
du	Diszk használat
df	Szabad diszkterület
quota	Diszk kvóta ellenőrzés

Óra 2.

- Shell alapelemek.

  változók,

  idézőjelek használata,

  parancssori paraméterek,

  aritmetikai műveletek

- Shell scriptek írása, végrehajtása.
- Shell scriptek engedély bitjei

Futtatása 1.

(complier ?????)

- sh *héjprogram neve*

```
echo „Hello World”
```

Futtatás 2.

- ./héjprogram neve
  (az adott könyvtárban lévőt futtatja)

„./hello.sh"

- /elérési út/héjprogram neve

„/home/pisti/bin/hello.sh"

- héjprogram neve
  (a PATH változóban megadott helyeken keresi és futtatja)
  PATH=„/bin:/usr/bin:/sbin:/usr/sbin:/home/pisti/bin:."

„hello.sh"

!!! FUTTATHATVÁ KELL TENNÜNK

„chmod" parancsal

```
#!/bin/sh
echo „Hello World"

vagy
-------------------------

#!/bin/sh
hello=„Hello World"
echo $hello

vagy
-------------------------

#!/bin/sh
# Én 1 megjegyzés vagyok és a
clear
# Kepernyo torlese
hello=„Hello World"
# Ertekadas
echo $hello
# Kiiratas
exit 0
# Vege
```

Chmod parancs (jogosultságok) engedélyek

*-rwx**rw-**r-- 1 halasza student 427 2005-12-14 12:55 program.sh*

chmod [augo][+-][rwx] fájlnév **. . .** (u - user, g - group, o - others, a – all)

x-futtatható (1), r-olvasható (4), w-írható (2)

chmod a+r filenev - olvasási jog mindenki számára
chmod +r filenev - ugyanaz, mivel az alapértelmezés az "all"
chmod go-rwx filenev - a tulajdonoson kívül senkinek semmi

chmod u+x filenev -- futtathatóvá teszi

chmod 100 filenev - csak futtatható lesz és csak a tulajdonosa számára
---x------ 1 halasza student 427 2005-12-14 12:55 program.sh

Ha elhagyjuk (hibák)

- Nem futtatható

ubi:~/script>program.sh

ubi:~/script>program.sh :Restricted acces

- Nincs benne a PATH ban

ubi:~/script>program.sh

ubi:~/script>program.sh :Command not found

Futás befolyásolása

- CTRL+C (megszakít)
- CTRL+D kilép (exit parancs)
- CTRL+Z (futás felfüggesztés)

jobs – megnézhetjük a háttérben futó folyamatokat

fg %n - újrakezd

Változók (környezeti változók)

- $HOME: login katalógus neve
- $PATH: keresési út a programok végrehajtásához
- $MAIL:a file neve ahova emailjeink érkeznek
- $PS1: elsődleges prompt
- $PS2: másodlagos prompt
- $EDITOR: alapértelmezett szövegszerkesztő
- $PWD: munkakönyvár (ahol éppen állunk)
- $OSTYPE:oprendszer típusa
- $SHELL: jelenleg futó shell típusa
- $TERM:terminál típusa (xterm,linux, vt100, stb)
- $RANDOM: véletlen szám
- $USER: felhasználói név
- $MAILCHECK: mailbox ellenőrzés gyakorisága
- Stb….. „man bash” „man sh”

Változók használata (szövegként kezel minden változót)

- szam=32
- szoveg1=hello
- szoveg2=world
- szoveg3=„hello world”

- ! (=) értékadásnál nincs [space] sem előtte sem utána
- ! Szöveget „” közzé tenni.

Változók használata 2. (kiiratás, értékátadás)

- szoveg1=hello
- szoveg2=world
- szoveg3=„hello world”

- echo „szoveg3”
- echo „$szoveg1 world”
- echo „$szoveg1 $szoveg2”

- szoveg4=$szoveg1” „ $szoveg2
- szoveg5=echo „$szoveg1 $szoveg2”

! Idézőjelek használata

@Milyen mértékig akarjuk betű szerint értelmezni@

- ‘…’ betű szerinti kiíratás (nincs értelmezés)

  echo ‘pontos ido: $ido’

- „…” a shell számára értelmes karaktereket értelmezi és behelyettesíti pl:$változó

  echo „pontos ido: $ido”

- `…` parancsvégrehajtás, behelyettesítés

  ido=`date`

!!!! A héj névvel nem rendelkező belső változói

$#	A parancssori paraméterek száma
$n	Az n-edik parancssori paraméter értéke (max 9)
$0	A pillanatnyi héjprogram neve
$$	A futó program azonosítója
$?	exit státusz
$-	A héjprogramot végrehajtó héjnak átadott kapcsolók
$*	Valamennyi parancssori paraméter egyben, egyetlen karakterláncként („$1 $2 ….$9)

program 1 2 5

echo $# = (3)

echo $3 = (5)

echo $0 = (program)

echo $* = (1 2 5)

Matematikai kifejezések

<, -lt	Kisebb
<=, -le	Kisebb egyenlő
=, -eq	egyenlő
==, -eq	Egyenlő
!=, -ne	Nem egyenlő
>=, -ge	Nagyobb egyenlő
>, -gt	Nagyobb
expr (reláció)	1 - igaz 0 – hamis

- \+ összeadás
- \- kivonás
- * szorzás
- / egészosztás
- % maradékképzés

@megkisérli számokká alakitani, ha nem akkor lexikális sorrendben hasonlitja össze a karaktereket@

- | „VAGY” operátor. Visszatérési értéke az első paraméter, ha az nem nulla, vagy nem üres karakterlánc, ellenkező esetben a második.
- & „ÉS” operátor. Visszatérési értéke az első paraméter, ha egyik argumentuma sem nulla vagy üres karakterlánc. Ellenkező esetben nulla.

expr (parancs)

Csak integerekben gondolkodik

szam=`expr 1 + 1`

szam=`expr 1 \* 1` (védő karakter)

echo $(($szam1+$szam2))

Feladat 1.

Paraméterként 2 db szám.

Dátum:xxxx
Felhasználói név: XXX
Programnév: xxxx

Elso szam:X
Masodik szam:y

összeg:xxxx
különbség:xxxx
szorzat:xxxx

hányados:
maradék:xxxx

Echo „szorzat `expr $szam1 \* $szam2` ”

Óra 3.

- Szokásos be és kimenetek, átirányítások, csövek, csatornák
- Adatfolyam szerkesztés
- Mintakeresés az állományokban, szűrők
- Hibacsatornák
- Beágyazott dokumentumok

- A parancsértelmező a parancsokat és adatokat a standard bementről vagy állományból olvassa.
- Unix terminológia: (háromágú átfolyó csatorna)

- szabványos bemenetre érkező adatokat fogadja *(ált. billentyűzet, …. file, másik program kimenete, hw eszköz,)*

(feldolgozza)

- szabványos kimentre küldi *(ált. képernyő… file, másik program bemenete, hw eszköz)*
- szabványos hibacsatorna (Hiba csatorna + egyéb közlendők) *(ált. képernyő, file)*

Shell I/O átirányítás

program > file (program kimenet file-ba, **ha már létezik felülírja !!!** ha nem létrehozza)

program >> file (kimenet file-hez való hozzáírása, ha nem létezik létrehozza)

program < file (program bemenetét file-ból veszi)

program << VEGE

ezt a program megkapja a

standard bemenetén a VEGE végjelig

VEGE

program1 | program2 (egyik program a másik bemenete)

program > /dev/audio

program > /dev/printer

program > /dev/null

Szűrők

- Tipikusan a szabványos inputról olvasnak és a szabványos kimenetre írnak. Egyszerűen csővezetékbe szervezhetők.
- Gyakran fájl paramétereket is értelmeznek (pl. more, cat, sort)

parancs1 | parancs2 –paraméter | parancs3 –paraméter .....

Néhány fontos szűrő

- cat — fájlok összemásolása
- more — fájl(ok) kiírása ernyőképenként
- less — fájl(ok) kiírása ernyőképenként
- head — fajl első n sora
- tail — fajl utolsó n sora
- tee — adatfolyam elágaztatása
- sort — sorbarendezés
- tr — karakter helyettesítő
- wc — sor és karakterszámláló
- diff — file összehasonlítás
- sed — adatfolyam editor
- **uniq** — előfordulást vizsgál

cat (szabványos bemenetről olvas szabványos bemenetre ír)

- cat (önmagában)
- cat adatok.txt
- cat > adatok.txt
- cat >> adatok.txt
- cat < adatok.txt

cat adatok.txt | sort

cat adatok.txt | sort >> rendezett_adatok.txt

more adatok.txt | sort >> rendezett_adatok.txt

less adatok.txt | grep „003642 | sort >> nyh-szamok-sorbarendezve.txt

Shell scriptekben

```
adatok=`cat adatok.txt`
echo $adatok
program $adatok
$adatok | program2
```

```
#!/bin/sh
telefonszamok=` less adatok.txt | grep –x „003642      "`
sikeres=`diald $telefonszamok | grep „modem detected"`
wardialer –u username.txt –pass dict.txt –num $sikeres
```

I/O átirányítás

A **>** **<** jelölések bármelyikét megelőzheti egy szám. Ekkor a szabványos bemenet ill. kimenet helyett a számnak megfelelő állományleírót kell érteni.

>&n a szabványos kimenet helyett az n. állományleírót használja

<&n a szabványos bemenet helyett az n. állományleírót használja

Csatornák számozása és a hibacsatornák átirányítása

- Szabványos bement (stdin) 0
- Szabványos kimenet (stdout) 1
- Szabványos hiba (stderr) 2

cat adatok.txt 1> kimenet.txt *(☺ nem jelöljük)*

cat adatok.txt 2> hiba.txt *(* 2>állomány a hibakimentet az állományba irányítja)

cat adatok.txt > kimenet.txt 2> hiba.txt

cat adatok.txt > kimenet+hibak.txt 2>&1 *(a* hibakimenet és a szabványos kimenet összekapcsolódik)

Saját hibaüzenet gyártása: *(programon belül)*

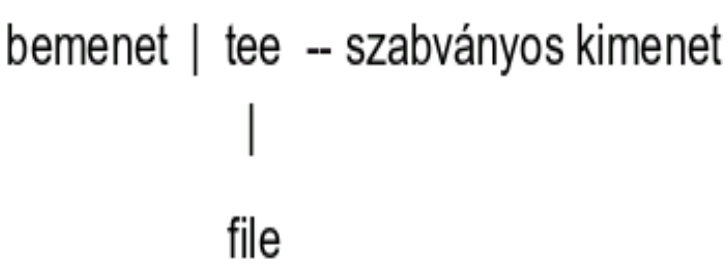
echo „Hibaüzenet ……..” *(nem jó mert csak a stdout-ra küldi)*

echo „Hibaüzenet ……..” 1>&2 (így már tudjuk stderr-ként kezelni)

☺ *Meg is szabadulhatunk a hibáktól:*

program 2> /dev/null

tee - (csövek elágaztatása)



ls -al | tee filelista

ps auxf | sort | tee processlista

Beágyazott dokumentum

cat << végjel

….szöveg

…

végjel

cat << számlavége

| *Előfizető neve* | *Telefonszáma* | *Számlaszám* | *Összeg* |

$felhasználó $telefonszam $számlaszám $osszeg

Dátum: $ido

számlavége

Óra 4.

- Programvezérlési szerkezetek.

  feltételek, ciklusok, test

Vezérlési szerkezetek - if

if [logikai kifejezés]
then
…parancsok…
elif [logikai kifejezés]
then …parancsok…
else
fi

test logikai kifejezés
vagy
[logikai kifejezés]

test = []

test Kifejezés kiértékelése. Visszatérési érték kijelző.

Fájlokra alkalmazott kapcsolók

test –r, -w, -x [file] *Értéke igaz, ha file létezik és* olvasható/írható/ futtatható

test –f [file] *Értéke igaz, ha file létezik.*

test –s [file] *Értéke igaz, ha file létezik és* hossza nem nulla

…

Karakterlánc kapcsolók

test $valt 1= $valt2 igaz, ha valt1 azonos valt2-vel

test –z $valt igaz ha valt hossza nulla

test –n $valt igaz ha valt hossza nem nulla

test $valt 1!= $valt2 igaz, ha valt1 nem azonos valt2-vel

Egész számokra alkalmazott kapcsolók

test n1 -le n2 igaz, ha n1 <= n2 (*-eq, -ne, -lt, -le, -gt, -ge)*

Logikai operátorok

! tagadás

-o, és

-a, vagy

(….) kiértékelési sorrend

```
if test $# -lt 1
then
echo Nincs paraméter!
exit
else
echo $*
fi
```

for

for változó **in** lista
do
parancsok
done

```
for i in  Hu Eu Com
do
echo $i
done



lista=`ls –al`
for i in $lista
do
echo $i
done
```

while

while [logikai kifejezés]
do
parancsok
done

```
i=1

while [ $i –le 10 ]
do


echo $i
i=`expr $i + 1`

done
```

until

until [logikai kifejezés]
do
parancsok
done

```
i=1

until [ $i –ge 10 ]
do


echo $i
i=`expr $i + 1`

done
```

többszörös elágazás

case változó **in**
    minta1) parancsok;;
    minta2) parancsok;;
    stb.
esac

```
read x

case $x in
a) echo Az 'a' betût nyomta le!;;
b) echo A 'b' betût nyomta le!;;
*) echo Egyéb betû!;;

esac
```

Óra 5

Függvények

```
fuggveny ()
{
Parancsok
}
```

fuggveny

fuggveny [parameterek]

```
fuggveny ()
{
 echo $valtozo
}



fuggveny ()
{
echo $1
}
```

Tudnivalók + hibák

Változói:

- (nem lokálisak) Hozzáférnek a főprogram változóihoz és maguk is létrehozhatnak újakat, ezeket a főprogram is örökölni ill. látni fogja.
- csak a függvény első meghívásakor jönnek létre

- önálló be és kimenettel, visszatérési értékkel, valamint saját parancssori paraméterekkel rendelkeznek. (!!!hibakezelés)
- visszatérési értéket működésük végén adják át

!!! Nem jó meghívás: fuggveny ()

Óra 6

Szabványos kifejezések (regexp regular expressions)

Mire valók

Alapelemek

Jelentésmódosító jelek

Példa

Mire való

Keresés, szűrés(karakterek,szövegrészeket)
cserék végrehajtása.

Szabályos kifejezésekkel vezérelhetőek a programok.

Mivel valósítjuk meg
grep,sed,awk

- A „grep” a bemenetre érkező sorok közül csak azokat küldi ki a kimenetre, amelyek megfelelnek a megadott kifejezésnek:

tail –f acces.log | grep „Login”

grep „Login” access.log

*grep „Login” **

- sed (stream editor) a megadott parancsok szerint módosítani is tudja a kimenetet (csere). Sor alapú de a teljes bemenetet egyben is kezelheti.

sed ‘s/Login:root/Login:nobody/’ -f access.log

- awk – programozási nyelv szövegfeldolgozásra. Soronként kezeli a bemenetet a kimenetet módosítani tudja. !!! A sorokon belül mezőket is megkülönböztet (lehetőséget ad a pozícionálásra).

Jan 2 09:00:01 zeus.nyf.hu Login: root

awk ‘{print $4,$6}’

zeus.nyf.hu root

Alapelemek

!!!! Kivételes karakterek *,.,[,],\,^,$,+ !!!!

Van valamilyen jelentésük

^ - *sor eleje* ***grep „^1” filename***

$ - *sor vége* ***grep „1$” filename***

. - *Tetszőleges karakter (kivéve újsor)* ***grep „...” filename***

\ - *speciális jelentés ki/(be) kapcsolás* ***grep „\.\.\.” filename***

[] – bármely karakter illeszkedése ***grep „[123]” filename***

- tartomány illeszkedése **grep „[1-3]” filename**
- nem illeszkedés **grep „[^1-3]” filename**

* - ismétlő karakter A * előtt álló karakter vagy kifejezés tetszőleges számú (akár 0) alőfordulása.

grep „A.*” filename

\+ - ismétlő karakter, de megköveteli, hogy az előtte lévő karakter/kifejezés legalább egyszer előforduljon.

grep „A\+” filename

() – csoportba rendezi a szabványos kifejezéseket, egységként kezelik

| - logikai VAGY

grep „\(A\|B\)” filename

{} – illeszkedések száma ({x} pontosan , {x,} legalább, {x,y} intervallum)

grep „A\{3\}” filename ,

grep „[1-9]\{10\}” fn, grep „[1-9]\{10,\}” fn, grep „[1-9]\{10,20\}” fn

- Példák:

„^[0-9]” - számmal kezdődik
„^[0-9]\{1\}[^0-9]” - 1 db számmal kezdődik követi valami
„^[0-9]\{1\}[^0-9]*” - 1 db számmal kezdődik, de lehet csak 1 db szám is

„Dr.\|dr.”
„Dr.[A-Za-z]\|dr.[A-Za-z]”
„Dr\.[A-Za-z]\|dr\.[A-Za-z]”
„Dr\.[A-Z][a-z]\|dr\.[A-Z][a-z]”
„Dr\.[ ][A-Z][a-z]|dr\.[ ][A-Z][a-z]”
„[dD]r\.[ ][A-Z][a-z]”
„^\([dD]r\.\[ ]){1,2}[ ][A-Z][a-z]\+”

Ora 7

A sed editor

- Működési elve
- Alapvető parancsai
- Kapcsolói

sed (stream editor)

- folymat editor vagy programozható szövegszerkesztő

A szabványos bemenetére érkező szöveget képes *(röptében)* feldolgozni és átalakítani (file. , pipe, stdin.)

- Hogyan: 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.

... | sed 'program'

sed 'program' filenév

- program = „a sed saját nyelvén írt szövegfeldolgozási utasítássorozatot jelenti (szabályos kifejezések + egybetűs kapcsolók”

Általános program: <cím1>,<cím2> parancs

- <cím>
- szám: (a bemenet adott sorszámú sora)

sed 10 parancs - csak a 10. soron hajtódik végre
sed 1,10 parancs - tartomány az 1-10 sorig hajtódik végre
sed parancs 10 - a keresett mintának csak a 10. előfordulásán hajtódik végre

(csak soron belül)

- szabványos kifejezés /...../

sed /[0-9]/ parancs - csak a számokat tartalmazó soron érvényesül
sed /[a-z]/ parancs - csak a kisbetűket tartalmazó soron érvényesül

Alapvető parancsai

- p kiíratás
- d törlés
- s helyettesítés
- a hozzáfűzés
- i beszúrás
- c a mintatér cseréje
- y a karakterek cseréje
- n next, még 1 sort olvas a bemenetről és hozzáfűzi a mintatérhez

p

cat szoveg | sed p - duplan jeleniti meg a sorokat

cat szoveg | sed '10 p' - duplan jeleniti meg a 10. a sort

cat szoveg | sed '1,10 p' - duplan jeleniti meg az 1-10-ig a sorokat

-n letilthajuk a default a default megjelenítést

cat szoveg | sed –n '10 p' - csak a 10. sort jeleniti meg

d

cat szoveg | sed '1,10 d' - törli a 1-10-ig a sorokat

s (g)

s/mit_cserélünk/mire cseréljük/ csak az első előfordulását
s/mit_cserélünk/mire cseréljük/g minden előfordulását
s/mit_cserélünk/mire cseréljük/n az n-ik előfordulását

echo 1234abba | sed ‘s/a/A/ ‘ 1234Abba
echo 1234abba | sed ‘s/a/A/g’ 1234AbbA

echo 1234abda | sed ‘s/[a-c]/A/g’ 1234AAdA

y

y/helyettesítendő/helyettesítő/ - a kettőnek karakterszámra egyenlőnek kell lennie

echo „abcdabcdABCDABCD” | sed ‘y/abcd/qxyz/ ’

Tippek

- sed ‘program’ szoveg.txt > szoveg.txt !!! NEM JOO

sed ‘program’ -i szoveg.txt

- utolsó sor $ szinbolum

sed –n ‘$p’ filename

Óra 9 Az AWK programnyelv

- Működésének alapelvei
- Az AWK nyelvi elemei
- Vezérlési szerkezetei

Alapelv

- Szövegfeldolgozásra szakosodott programnyelv, több változat: awk, gawk (GNU project), tawk (MS-Windows DLL), awka,mksawk,awkcc (c interpreter) , mawk, original-awk
- A C szövegszerkesztésre kihegyezett változata, képességei azonosak egy grafikus felületen futó táblázatkezelővel
- Az awk alapvető feladata, hogy olyan szövegegységeket keressen file-okban, amelyek tartalmaznak egy bizonyos mintát.
- A minta lehet szabványos kifejezés vagy logikai feltétel, ha nincs minta minden sorra végrehajtja
- Az awk programok adatvezéreltek (először az adatot adjuk meg amivel dolgozni szeretnének aztán azt, hogy mit szeretnénk csinálni)

Futtatása

- **Szabványos bemenetről:**

  … | awk '{parancsok}'

- **Feldolgozandó file megadásával:**

  awk '{parancsok}' filename1 <filename2>

- **Ha a program hosszabb vagy külön file-ban van:**

  … | awk –f programfile

  awk –f programfile filename1 <filename2>

programfile.awk

#!/bin/awk

Feldolgozás

minta1 {tevékenység1}
minta2 {tevékenység2}

- Soronként történik, a sorokat mezőkre osztja (legkisebb feldolgozási egység)
- A mezőket a mezőelválasztó karakterrel tagolhatjuk, értéke definiálható [-F] (alapértelmezetten: szóköz, tabulátor)

  !!! (LEHET SZABVÁNYOS KIFEJEZÉS IS)

  hivatkozás: $1,$2…..$i, a $0 az egész sort jelöli
- A minták megadása reguláris kifejezéssel /…/ jelek között,

  logikai feltételek a C-ben megszokott operátorokkal „a>2”

- Kiíratás:print, printf*** (újsor karakter nélkül)

  awk ‘{print $2,$1}’ ; awk ‘{print $2” „$1}’ ; awk ‘{print Mezők:$2” „$1}’ ;

Változók

- Felhasználó által megadott változók:

  I. nem kell külön deklarálni, első értékadáskor létrejönnek

  II. Megadhatjuk program futása előtt is, vagy a héjprogram változóját is átvehetjük.

less /etc/passwd | awk -F: '{nev="Falhaszaloi nev:" ; konyvt="Konyvtar: " ; print nev,$1" "konyvt$6}'

less /etc/passwd | awk -F: -v nev="Falhaszaloi nev:" -v konyvt="Konyvtar:" '{print nev,$1" "konyvt$6}'

III. BEGIN blokk használata:

awk –F: 'BEGIN {nev="Falhaszaloi nev:" ; konyvt="Konyvtar:"} {print nev,$1", "konyvt$6}'

Változók II.

- Belső változók:
  - FILENAME aktuális bemeneti fájl neve
  - FS bemeneti mezőelválasztó karakter
  - NF az aktuális sor mezőinek száma
  - NR az aktuális sor száma
  - OFS kimeneti mezőelválasztó karakter

awk -v FS=: -v nev="Falhaszaloi nev:" -v konyvt="Konyvtar:" '{print nev,$1", "konyvt$6}'

Megszámozott sorok:

awk -v FS=: -v nev="Falhaszaloi nev:" -v konyvt="Konyvtar:" '{print NR,"„,nev,$1", "konyvt$6}'

A BEGIN , END blokk

- A főciklustól független műveletek végrehajtását teszi lehetővé
- A BEGIN még a sorok feldolgozása előtt, de csak egyszer fog végrehajtani
- Az END a sorok feldolgozása után végez el valamilyen műveleteket

awk 'BEGIN { bevezető műveletek} {főprogram} END {záró műveletek}'

awk -F: 'BEGIN {nev="Falhaszaloi nev:" ; konyvt="Konyvtar:" ; print "ELEJE"} {print nev,$1", "konyvt$6} END {print "VEGE"}'

Operátorok

= értékadás

|| vagy

&& és

! tagadás

>= < <= == != relációs operátorok

+ - * / % ++ -- aritmetikai operátorok

Vezérlési szerkezetek (ua. mint „C””

```
if(feltétel)
 utasítás
 else
 utasítás

for(kifejezés1;feltétel;kifejezés2)
 utasítás

for(változó in tömb) utasítás

while(feltétel)
 utasítás
 break
 continue
 next
 exit
```

```
{if ($2!==root) print $2}

{for (i=1;i<11;i++)}

{i=0
while (i<10)
print i
i++}
```

Beépített függvények

(a különböző változatoknál eltérhetnek)

. cos(kif) kif koszinusza

· exp(kif) kif exponenciális függvénye

· getline() következő sor beolvasása. Visszatérés: 0, ha fájl vége, 1 ha nem

· index(s1,s2) s2 kezdőpozíciója s1-ben

· int(kif) kif egészrésze

· length(s) s string hossza

· log(kif) kif logaritmusa

· sin(kif) kif szinusza

· split(s,a,d) s-t d elválasztójel szerint a[1]...a[n] tömbelemekre osztja, visszatérési értéke n

· sprintf(fmt,...) a ...-ot fmt formátum string szerint formázva adja vissza

· substr(s,m,n) s string m-edik karaktertől kezdődő n karakteres része

A printf használata

A kiíratás vezérlése a “%”-jellel történik:

('%' jelhez tartozik egy sorszámozott paraméter, az érték megjelenítését egy betű jelöli)

- szám esetén: "%d" decimális, "%x" hexadecimális
- sztringekkel esetén: %s-t használunk

A kiírás szélességét a '%' jel után lehet megadni, ha az adott szöveg ennél rövidebb, akkor előről szóközökkel lesz feltöltve (jobbra zárás), azonban ha a '%' jel után egy '-' áll, akkor balra zárt.

awk ‘{printf „%10s”,$1}’