CSILLAGÁSZATI LABORATÓRIUM I.

2. Az awk alapjai

Az `awk` egy nagyon erőteljes, sokmindenre használható parancs, melynek segítségével képesek vagyunk adattömböket tartalmazó fájlokkal különböző műveleteket végrehajtani. Segítségével unix parancsokkal babrálhatunk. Használatával az *IRAF* adatbázisok rendezése, redukciója lényegesen egyszerűsödik.

A jegyzet alján sok példa script található, melynek segítségével az `awk` könnyen elsajátítható alapszinten. Az egyes scriptek nagyon megkönnyebíthetik az ember dolgát, ha nagy mennyiségű adattal van dolga.

Az `awk`-ot kétféleképpen futtathatjuk. Egyszerűbben a terminál parancssorából, mintha csak egy shell scriptet futtatnánk. A kicsit komolyabb `awk` scripteket már fájlból futtatjuk. Mindkettőre található példa a jegyzet végén.

Az `awk` a bemenet minden sorát megnézi, és megpróbálja azonosítani a megadott „mintát", és a `{}` jeleken belüli műveleteket elvégzi a mintára. Később ez a zagyvaság érthető lesz. Számunkra a lefontosabb az `awk`-nak az a tulajdonsága, hogy nagyon jól kezeli le az oszlopokkal és sorokkal jellemezhető adattömböket tartalmazó fájlokat. Az egyes adatokat üres hellyel (tabokkal vagy space-ekkel) kell elválasztani. Ha netán az adatokat valamilyen karakter választja el egymástól, akkor a `-F` kapcsolóval kikerülhetjük az ebből adódó hibákat.

Az `awk` az első oszlopra `$1`-gyel, a másodikra `$2`-vel . . . stb hivatkozik. Az egész adatsorra pedig `$0`-val.

További magyarázatok helyett megpróbálom példákon át szemléltetni a működését. Legyen egy fájlunk: `file1` melyben legyen kettő, számokat tartalmazó oszlopunk. Célunk létrehozni egy új fájlt: `file2` melyben megtartjuk az első két oszlopot és létrehozunk egy harmadik oszlopot, mely az első két oszlopban lévő számok arányát tartalmazza. Ha csak azokat a sorokat akarjuk megtartani, melyre igaz, hogy az első oszlopban lévő szám kisebb, mint a második oszlopban lévő szám, akkor a következő sorok valamelyikét kell a shell parancssorába begépelnünk:

```
$ awk '$1 < $2 {print $0, $1/$2}' file1 > file2
```

vagy

```
$ cat file1 | awk '$1 < $2 {print $0, $1/$2}' > file2
```

Értelmezzük egy kicsit ezt a második parancssort. Mint tudjuk a linux-os jegyzetből, a `cat file1` parancs a `file1` egész tartalmát a képernyőre írja ki. A „pipe" jel (`|`) ezt a kimenetet a képernyő helyett az `awk` parancsba irányítja át. A program a bemenetet soronként értelmezi és ugyanúgy, mint az első példánál is, próbálja a megadott „mintát" megkeresni. A minta bármi lehet az első `'` jel és a a `{` között. Ez esetben a minta: `$1 < $2`. Ha a mintát nem tudja illeszteni, akkor a következő sorra megy. Ha viszont teljesül, akkor végrehatja a parancsot, mely a `{}` jelek közt található. Ha nem adunk meg semmi mintát, akkor rögtön végrehatja az utasjtást ami a `{}` jeleken belül található.

Esetünkben a végrehajtandó feladat általában valaminek a kiiratása. Ezt a `print` paranccsal hajtjuk végre. Ez esetben azt szeretnénk, ha az egész sort kiírná (`$0`) és utána pedig az első két oszlop arányát (`$1/$2`). A feladatot a `}` jellel zárjuk. Magát az `awk` parancsot pedig a `'` jellel. A kimenetet a `file2`-be irányítjuk át (különben a standard kimenetre, azaz a képernyőre írná ki).

A következő példában egy gyakran előforduló probléma megoldását mutatom be. Tegyük fel, hogy az *IRAF*-es adatfeldolgozás eredményeképpen több ezer adatfájlunk van és ezeket szeretnénk egy `.dat` kiterjesztéssel ellátni és egy újabb mappába áthelyezni. Ezt lehet egyesével is megtenni természetesen, de akkor készüljünk fel sok-sok élelemmel. Az `mc` segítségével pár perc alatt megoldhatjuk a problémát, viszont az `awk`-kal pár másodperc alatt.

Legyenek a fájlaink neveinek kezdete `ngc7538_` (a `*` és a `?` itt is ugyanúgy használható, mint a shellben). Az adatokat át szeretnénk helyezni a `../iraf` könyvtárba is az átnevezés mellett. Ehhez a következő parancssort kell begépelnünk

```
$ ls ngc7538_* | awk '{print "mv "$0" ../iraf/"$0".dat"}' | bash
```

Értelmezzük ezt a parancssort. Az `ls ngc7538_*` kiiratja a standard kimenetre (képernyő) az ilyen kezdettel jellemezhető fájlok neveit, soronként. Ezt a kimenetet mi átirányítjuk az `awk`-ba. Itt láthatóan mintát nem adtunk meg, azaz a parancs végrehajtódik minden sorra. Például ha az `ls ngc7538_*` parancsnak az első két sorában az szerepel, hogy `ngc7538_0001r` illetve `ngc7538_0002r`, akkor az `awk` a következőt írná ki:

```
mv ngc7538_0001r ../iraf/ngc7538_0001r.dat
mv ngc7538_0002r ../iraf/ngc7538_0002r.dat
```

Látható, hogy az `mv` parancsot a kimenetre adja meg az `awk` alapból. Ezt tovább kell irányítanunk a parancsértelmezőnek, a *bash shell*-nek. Ezt tesszük a `| bash` „pipe"-olással.

A bonyolultabb `awk` scripteket már fájlba szokás írni. Ezeket a fájlokat a következő módon futtathatjuk:

```
$ cat file1 | awk -f a.awk > file2
```

Itt a `file1` a bemenő fájl, míg `file2` a kimenő fájl. Az `a.awk` az `awk` parancsokat tartalmazó fájl. Azokat a parancsokat, melyek több sorosak fájlból kell futtatni.

Van egy-két előre definiált `awk` változó. Ezek a következők:

- `NF` - az oszlopok száma
- `NR` - az aktuális sornak a száma, mellyel dolgozik
- `END` - akkor válik igazzá, ha az `awk` eléri a fájl végét
- `BEGIN` - a parancsok végrehajtása előtt igaz
- `length` - a karakterek számát adja meg egy sorban vagy stringben

Ezenek kívül az `awk` tud keresni is a (/) jellel, illetve stringeken belül is tud keresni. Vannak logikai kapcsolók is, például: `||` (vagy), `&&` (és). Ezeket a minta megadásban tudjuk használni. Lehetőség van ezeken kívül saját változók definiálására is. Létezik az `awk`-nak egy újabb verziója, a `gawk`.

Példák

Az `awk` scriptekben kommentelni a # jellel lehet. Cseréljük fel egy fájl első két oszlopát, és irassuk ki a képernyőre az első két oszlopot

```
$ awk '{ print $2, $1 }' file
```

Irassuk ki azokat a sorokat a képernyőre egy fájlból, melyek hosszabbak 72 karakternél.

```
$ awk 'length > 72 {print $0}' file
```

Irassuk ki a második oszlopban lévő stringek hosszát a képernyőre.

```
$ awk '{print length($2)}' file
```

Adjuk össze az első oszlopban szereplő értékeket, majd irassuk ki az összeg értékét, illetve az egyes értékek átlagát. Ezt már `awk` scriptként kell lementeni.

```
    { s += $1 }
END { print "az osszeg:", s, " az atlag:", s/NR }
```

Az oszlopokat fordított sorrendben irassuk ki!

```
$ awk '{ for (i = NF; i > 0; --i) print $i }' file
```

Irassuk ki az utolsó sort (ez szintén scriptes)!

```
    {line = $0}
END {print line}
```

Irassuk ki azon sorok számát, melyekben megtalálható a „macska” szó.

```
/macska/ {nlines = nlines + 1}
END {print nlines}
```

Az összes olyan sort írja ki, melynek az első oszlopa különbözik az előtte lévő sor első oszlopától.

```
$ awk '$1 != prev {print $0; prev = $1}' file
```

Írja ki a harmadik oszlopot, ha az első oszlop értéke nagyobb a másodikénál.

```
$ awk '$1 > $2 {print $3}' file
```

Írja ki a sort, ha a harmadik oszlopban szereplő érték nagyobb a második sorban szereplő értéknél

```
$ awk '$3 > $2 {print $0}' file
```

Írja ki, hogy összesen hány sornál teljesül az, hogy a harmadik oszlop értéke nagyobb az első oszlop értékénél.

```
$ awk '$3 > $1 {print i + "1"; i++}' file
```

Írja ki a sor számát, majd az első oszlop értékét

```
$ awk '{print NR, $1}' file
```

Írja ki az összes oszlopot, kivéve a másodikat

```
$ awk '{$2 = ""; print $0 }' file
```

Írja ki azt, hogy Szia tízszer

```
$ yes | head -28 | awk '{ print "Szia"}'
```

Írja ki azt, hogy hi.0010 egészen hi.0099-ig

```
$ yes | head -90 | awk '{printf("hi.00%2.0f \n", NR+9)}'
```

Minden értékhez az abszolútértékét írja

```
$ { for (i = 1; i <= NF; i=i+1) if ($i < 0) $i = -$i print}
```

Ha van olyan karakter, melyet nem akarunk az oszlopok közé venni, akkor a -F kapcsolót kell használni. Legyen az alábbi adatfájlunk:

```
000902|Beavis|Theodore|333-242-2222|149092
000901|Jones|Bill|532-382-0342|234023
...
```

Ebből akarjuk „Jones”-nak a telefonszámát kibányászni. Ehhez a következő parancssort kell begépelnünk:

```
$ awk -F"|" '$2=="Jones"{print $4}' filename
```

Bizonyos sorszámú nyomtatási feladatok törléséhez:

```
BEGIN{
      for (i=875;i>833;i-){
              printf "lprm -Plw %d\n", i
      } exit
     }
```

0, 1-enként binneljük a második oszlopot 0 és 1 között

```
$2 <= 0.1 {na=na+1}
($2 > 0.1) && ($2 <= 0.2) {nb = nb+1}
($2 > 0.2) && ($2 <= 0.3) {nc = nc+1}
($2 > 0.3) && ($2 <= 0.4) {nd = nd+1}
($2 > 0.4) && ($2 <= 0.5) {ne = ne+1}
($2 > 0.5) && ($2 <= 0.6) {nf = nf+1}
($2 > 0.6) && ($2 <= 0.7) {ng = ng+1}
($2 > 0.7) && ($2 <= 0.8) {nh = nh+1}
($2 > 0.8) && ($2 <= 0.9) {ni = ni+1}
($2 > 0.9) {nj = nj+1}
END {print na, nb, nc, nd, ne, nf, ng, nh, ni, nj, NR}
```

Minimum és maximum megkeresése az első oszlop elemei között

```
NR == 1 {m=$1 ; p=$1}
$1 >= m {m = $1}
$1 <= p {p = $1}
END { print "Max = " m, " Min = " p }
```