SED

*Stream editörün kısaltmasıdır.

*Ed line editörden türetilmiştir.

*Sed ‘3,4p’ foo.txt // İlgili verilen akış üzerinde verdiğimiz aralığın duplicate(tekrardan yazılması) etmeyi sağlıyor. (tek parametre ile çalışmıyor)

*sed ‘4q’ foo.txt dosyanın başından itibaren verilen satıra kadar yazdırma işlemi yapar ve geri kalanı yazdırılmaz. Mesela 4. Satırda basmayı bırakacak bu kod.

* sed -n ‘3,4p’ foo.txt sadece verilen aralığın gösterilmesini sağlar. Sadece ilgili kısmı getiriyor.

*sed -n ‘$p’ foo.txt ilgili dosyanın sadece son satırını getirdi.

*sed -n ‘3,$!p’ foo.txt verilen aralıktaki satırların dışındaki satırları getirir. 3. Satırdan sonrakileri getirir.

NOT: kısacası -n verirsek bastırıyor, -n yazmazsak duplicate yapıyor.

SED DÜZENLİ İFADELER İLE KULLANIMI

*sed -n ‘/^ FROM: /p’ $HOME/mbox “$HOME/mbox” dosyası üzerinde “ From:” (<BOŞLUK> From:) ile başlayan satırları bana göster diyoruz.

Ls -l | sed -n ‘/^…..w/p’ Başlangıçtan itibaren 5 karakter ne olursa olabilir 6. karakter “w” olsun. Ls çıktısında 1. Değer dosya tipiydi, diğerleri 3er ugo. Yani aslında grubun yazma hakkın sahip dosyalarını listelemiş olduk.

Sed:Substition DEĞİŞTİRME İŞLEMİ

Sed ‘s/|/:/g’ data.txt burada slashlar arasında kalan karakterleri değiştirir. Yani her bulduğu “/” için “:” koyar. Buradaki “g” aramanın sürekli yapılacağı anlamına geliyor. Yani işlemi yaptıktan sonra bir sonraki işlem kaldığımız yerden devam eder. “g” yi vermezsek en baştan tekrar arar

---

AWK

Aho, Weinberger ve Kerninga oluşturan kişiler ve baş harflerinden oluşuyor.

Awk -f awk.script foo.txt

Begin içerisine, belirtilen dosyayı okuma işlemine geçmeden önce yapmak istediğimiz işlemleri buraya yazıyoruz.

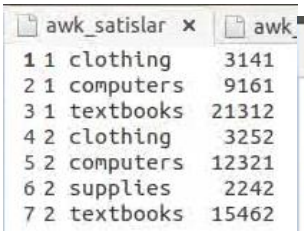
Ortadaki süslü parantezler içerisinde ise, alınan dosyanın HER BİR SATIRI için gerçekleşecek işlemleri yapıyoruz.

End içerisinde ise Okuma işlemi bittikten sonra gerçekleştirilecek işlemlerin belirtildiği yerdir.

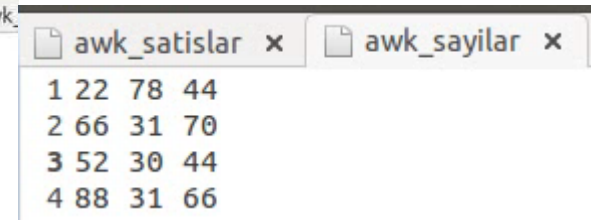
Örnek:

```
# Begin Processing
BEGIN {print "Print Totals"}

# Body Processing
{total = $1 + $2 + $3}
{print $1 " + " $2 " + " $3 " = "total}

# End Processing
END {print "End Totals"}
```

Yukarıdaki örnekte dolarlar ile belirtilen yerler sütunlara tekabül eder.

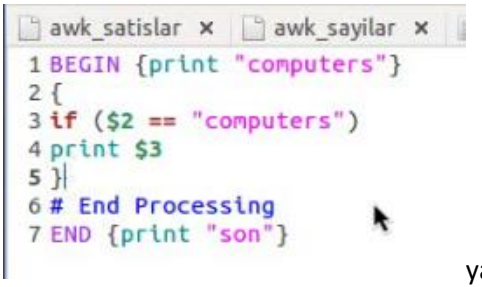


```
1 1 clothing    3141
2 1 computers   9161
3 1 textbooks  21312
4 2 clothing    3252
5 2 computers  12321
6 2 supplies    2242
7 2 textbooks  15462
```



```
1 22 78 44
2 66 31 70
3 52 30 44
4 88 31 66
```

Yukarıdaki kod 1. 2. Ve 3. Sütunları toplayarak total e atıyor ve onu print ediyor.

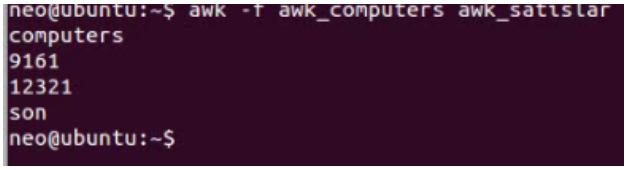


```
1 BEGIN {print "computers"}
2 {
3 if ($2 == "computers")
4 print $3
5 }
6 # End Processing
7 END {print "son"}
```

yandaki kod parçasında 2. Sütundaki her eleman computer mi diye kontrol ediyor, eğer öyle ise gidip fiyatını yazdırıyor. Tüm computerler bitince de son yazdırıyor.

Yukarıdaki kodu koşmak için;

Awk -f awk_computers awk_satislar yazıyoruz. Çıktı:



```
neo@ubuntu:~$ awk -f awk_computers awk_satislar
computers
9161
12321
son
neo@ubuntu:~$
```

AWK Döngüleri

awk_satislar × awk_sayilar × awk_satislari_toplama ×

```
1 BEGIN {OFS = "\t"}
2 {deptSales [$2] += $3}
3 END {for (item in deptSales)
4         {
5         print item, ":", deptSales[item]
6         totalSales += deptSales[item]
7         } # for
8         print "Total Sales", ":", totalSales
9 } # END
```

Yukarıdaki kod parçasında OFS bir çevre değişkenidir ve bu değerin default değeri tek bir boşluktur, eğer biz bunu değiştirirsek yazdırma anında elemanlar arasındaki karakteri değiştirmiş oluyoruz. Yani kısaca basma işleminde kullanılacak işlem boşlul değil, “TAB” olacağı için (\ t) tab bırakacak aralara. Sütunsal bazda hizalama yapmak için.

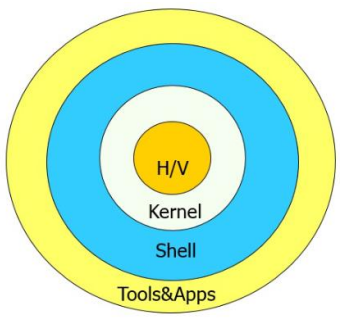
Her bir turda yani ilk satırda icra edilecek kısım deptSales[$2], gidip clothingi alıyor sonra değerini topluyor, 2. Adımda gidip computersi topluyor, yani aslında deptSales[computers] gibi bir indisin içerisine değer atıyor. Böyle bir durumda Computers için listedeki tüm computersleri toplayarak tek bir index içinde toplam değerlerini toplamış oluyor.

For döngüsü içerisinde itemleri alıyor, aldığı itemleri yazdırıyor, sonrasında itemin bulunduğu indexi diziye verip o birimin toplam satışını görmüş oluyoruz. Total sales içerisinde ise bütün birimlerin kendi değerlerini tekrar toplayarak totaldeki satışı oluşturuyor, for bitince de toplam yapılan bütün genel satış değerini yazdırıyor.

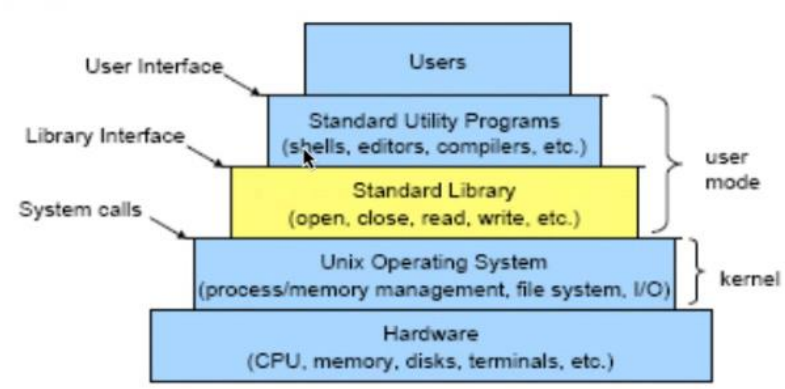
Çıktı:

```
neo@ubuntu: ~
neo@ubuntu:~$ awk -f awk_satislari_toplama awk_satislar
supplies        :       2242
computers       :       21482
textbooks       :       36774
clothing        :       6393
Total Sales     :       66891
neo@ubuntu:~$
```

FILE MANAGEMENT



Unix tabanlı sistemlerin katmanları:



UNIX SİSTEM PROGRAMLAMA

Programlar sistem çağrılarını kullanarak kütüphanelere erişirler.

Sistem çağrı tipleri:

*File I/O dosya giriş çıkışları

*Process Management process yönetimi

*Inter Process Communication(IPC)

*Signal Handling

C dili vs C++

C de string tipi diye bişey yok.

C için;

*Strcpy() kopyalıyor

*Strcmp() karşılaştırma yapıyor.

C++ de printf ve cout var

C de scanf ve fgets() var

BASIT DOSYA GİRİŞ ÇIKIŞLARI I/O

Dosyalar okuma veya yazma modunda açılabiliyor.

Bu sistem çağrılarını kullanabilmek için <stdio.h> (Standard input output kısaltması) incude etmemiz lazım

Her bir dosya bir dosya işaretçisi tarafından referanslanır.

3 tane dosya otomatik olarak açılır:

FD 0 standart input

FD1 standart output

FD2 standart error

Open() FONKSİYONU

- `fd = open(path, flags, mode);`
- `path`: `char*`, absolute or relative path
- `flags`:
  - O_RDONLY – open for reading
  - O_WRONLY – open for writing
  - O_RDWR – open for reading and writing
  - O_CREAT – create the file if it doesn't exist
  - O_TRUNC – truncate the file if it exists (overwrite)
  - O_APPEND – only write at the end of the file
- `mode`: specify permissions if using O_CREAT
- Returns newly assigned file descriptor
- `fd = open("myfile", O_CREAT, 00644)`

Open içindeki flag dediği şey dosy tipidir. Mode hangi modda açılacağı. Path ise dosyanın yolunu verir.

O_RDONLY ylnızca okunur

O_WRONLY sadece yaılabilir.

O_RDWR okunabilir ve yazılabilir.

O_CREAT öyle bir dosya yoksa oluşturur

O_TRUNC dosya varsa içini boşaltır

O_APPEND dosyanın sonuna ekler.

Fd= open("myfile", O_CREAT, 00644) burada program koşunca myfile diye dosya oluşturacak sürekli.

Red() FONKSİYONU

- `bytes = read(fd, buffer, count);`
- Read from file associated with `fd`; place `count` bytes into `buffer`
- `fd`: file descriptor to read from
- `buffer`: pointer to array of `count` bytes
- `count`: maximum number of bytes to read
- Returns the number of bytes read or `-1` on error

```
int fd = open("someFile", O_RDONLY);
char buffer[4];
int bytes =
  read(fd, buffer, 4);
```

1.parametresi Okuma yapacağımız dosya tanımlayıcısı.

2.parametresi Okunacak verilerin aktarılacağı dizidir.(buffer)

3.parametresi Dosyadan kaç tane veri okuyacağımızı gireriz. (genelde buffer kadar beklenir ama daha az da olabilir.)

Write() FONKSİYONU

- `bytes = write(fd,buffer, count)`
- Write contents of `buffer` to a file associated with `fd`
- `fd`: file descriptor
- `buffer`: pointer to an array of `count` bytes
- `count`: number of bytes to write
- Returns the number of bytes written or `-1` on error

```
int fd = open("someFile", O_WRONLY);
char buffer[4];
int bytes =
  write(fd, buffer, 4);
```

Close() fonksiyonu ile işleminiz bitince kapatabilirsiniz dosyayı.

Lseek() fonksiyonu ile arama işlemleri yapılır.

lseek()

- `retval = lseek(fd,offset,whence)`
- Move file pointer to new location
- `fd`: file descriptor
- `offset`: number of bytes
- `whence`: .
  - `SEEK_SET` – offset from beginning of file
  - `SEEK_CUR` – offset from current offset location
  - `SEEK_END` – offset from end of file
- Returns offset from beginning of file or –1 on error

2. parametre Offset değeri dosya içinde nerede olduğumuz yer.

3. parametre offset değerinin baz alınacağı durum.

SEEK_SET Dosyanın başını gösteriyor

SEEK_CUR aradığımız veriyi bulunca kaldığımız yerden devam edebilmemiz için o konumda CUR kullanılıyor.(recursive bi mantıkla dosya sonuna kadar gidebilmek için)

SEEL_END aynı şekilde CUR gibi ancak sondan arama yapıyor.

fopen()

- `FILE *file_stream = fopen(path, mode);`
- `path`: `char*`, absolute or relative path
- `mode`:
  - `r` – open file for reading
  - `r+` – open file for reading and writing
  - `w` – overwrite file or create file for writing
  - `w+` – open for reading and writing; overwrites file
  - `a` – open file for appending (writing at end of file)
  - `a+` – open file for appending and reading
- `fclose(file_stream);`
  - Closes open file stream

*r dosyayı okuma modu

*r+ okuma yazma modu

*w dosyayı yazma modunda açıyoruz, dosya yoksada oluşturur

*w+ modunda açarsak dosyayı hem okuma hem yazma modunda açıyoruz, eğer dosya varsa da içini boşaltıyoruz.

*a dosyayı sondan eklemek için açıyor

*a+ dosyayı hem sondan eklemek hemde okumak amacıyla açıyor.

Fclose ile içine verilen dosyayı kapatıyor kod içinde.